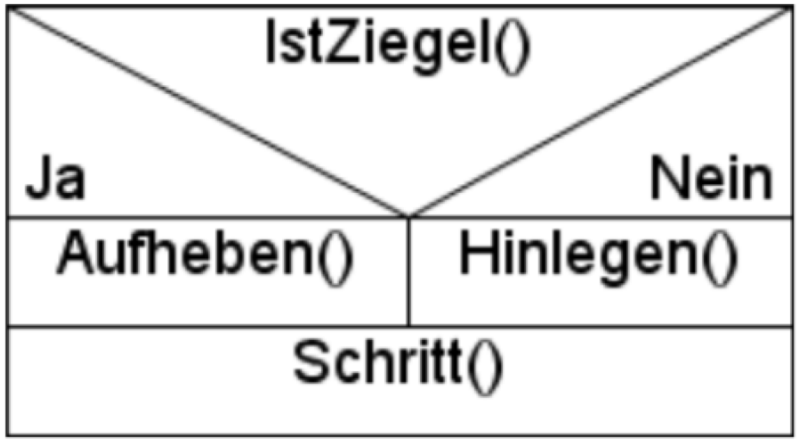
3. Bedingte Anweisungen

Fallunterscheidungen der Form WENN...DANN... in der Informatik kennst du aus der 7. Klasse beim Programmieren mit Karol sowie aus der 9. Klasse beim Arbeiten mit Tabellen und Datenbanken.

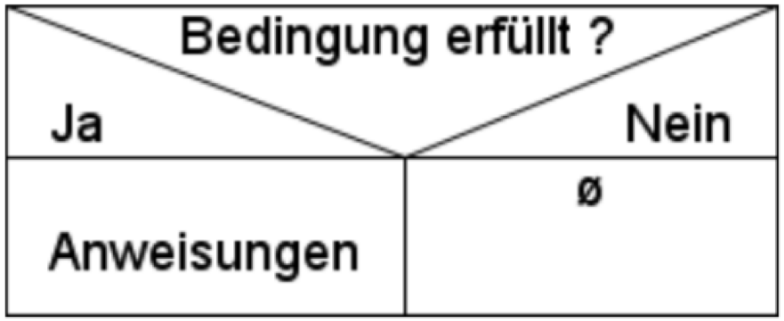
Grafisch kann man sie als Struktogramm darstellen.

Beispiel:



MERKE

Einfache bedingte Anweisung




```
if ( Bedingung ) {
   Anweisung;
}
```

Die Bedingung ist ein Ausdruck oder ein Methodenaufruf, der den Wert „true“ oder „false“ zurückgibt.
Beispiele :

```
if ( a < b )  { a = a + 1; }

if ( name == “Richtig“ )  { return name;}

if ( karol.IstZiegel() )  { karol.Aufheben() };
```

MERKE

Bedingte Anweisung mit Alternative

Bedingung erfüllt ?	
Ja	Nein
Anweisungen	andere Anweisungen

```
if ( Bedingung ) {
   Anweisung1;
}
else {
   Anweisung2;
}
```

Beispiel :

```
if ( a < b )  {
  a = a + 1;
}
else {
  a = a – 1;
}
```

MERKE

Mehrfache Auswahl

MEHRFACHE FALLUNTERSCHEIDUNG			
Bedingung			
Fall 1	Fall 2	Fall 3	sonst
Anweisung 1	Anweisung 2	Anweisung 3	∅

```
if  ( Fall 1 ) {
    Anweisung 1;
}
else if ( Fall 2 ) {
    Anweisung 2;
}
else if ( Fall 3 ) {
    Anweisung 3;
}
```

MERKE

Mehrfache Auswahl mit Alternative

MEHRFACHE FALLUNTERSCHEIDUNG			
Bedingung			
Fall 1	Fall 2	Fall 3	sonst
Anweisung 1	Anweisung 2	Anweisung 3	Anweisung 4

```
if  ( Fall 1 ) {
    Anweisung 1;
}
else if ( Fall 2 ) {
    Anweisung 2;
}
else if ( Fall 3 ) {
    Anweisung 3;
}
else {
 Anweisung 4;
}
```

MERKE

Mehrfache Auswahl mit Alternative

Beispiel :

```
if ( tag == “Montag“) {
  b = 1;
}
else if ( tag == “Dienstag“) {
  b = 2;
}
else if ( tag == “Mittwoch“) {
  b = 3;
}
else {
  b = -1;
}
```

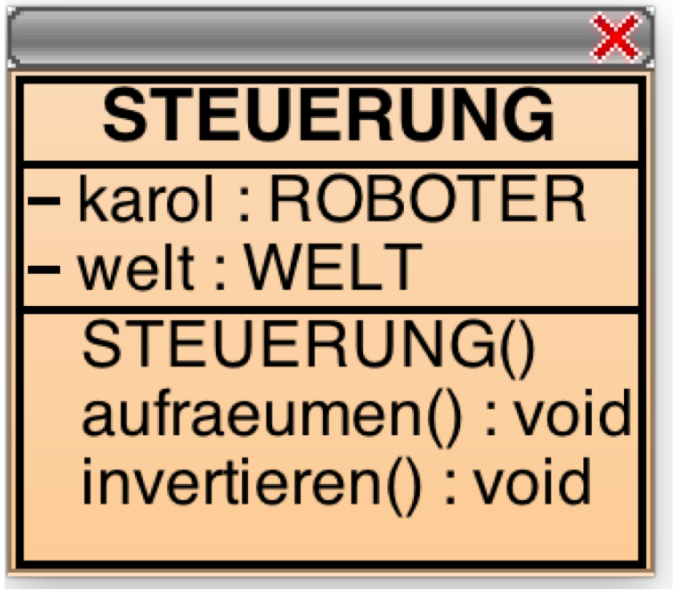
MEHRFACHE FALLUNTERSCHEIDUNG			
Bedingung			
Fall 1	Fall 2	Fall 3	sonst
Anweisung 1	Anweisung 2	Anweisung 3	Anweisung 4

Übung 1 – JAVA Karol

a)
Kopiere das BlueJ-Projekt „JavaKarol“
in deinen Ordner und ändere
den Namen im „JavaKarolBedingungen “ ab.

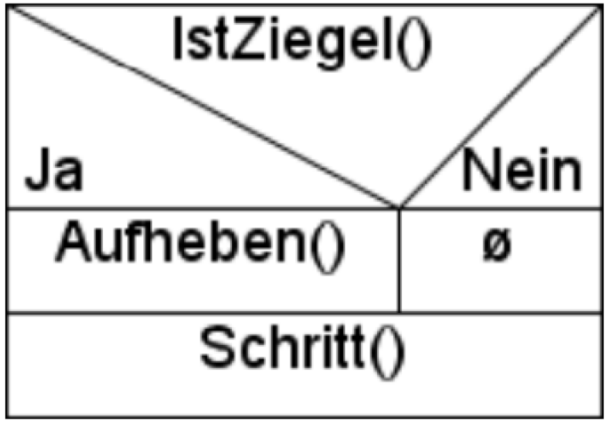
b)
Erzeuge darin eine neue Klasse STEUERUNG
nach nebenstehender Klassenkarte.
Ein – vor einem Attribut bedeutet *private.*

Die Welt wird im Quelltext durch den Befehl
welt = new WELT("welt-01.kdw");
erzeugt.



Übung 1 – JAVA Karol

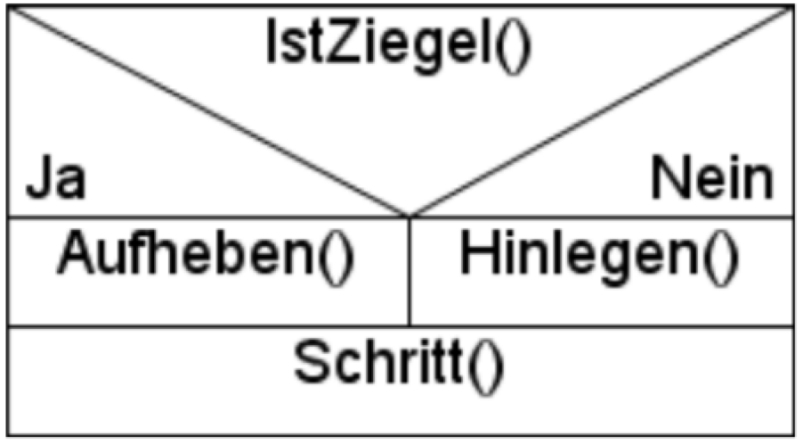
c)
Schreibe die Methode *aufrauemen()* und teste sie.




```
public void aufraeumen() {
    if( karol.IstZiegel() ) {
        karol.Aufheben() ;
    }
    karol.Schritt();
}
```

Übung 1 – JAVA Karol

d)
Schreibe die Methode invertieren() und teste sie.
Verwende wie im Struktogramm vorgegeben die bedingte Anweisung mit Alternative.
(*if* und *else*)

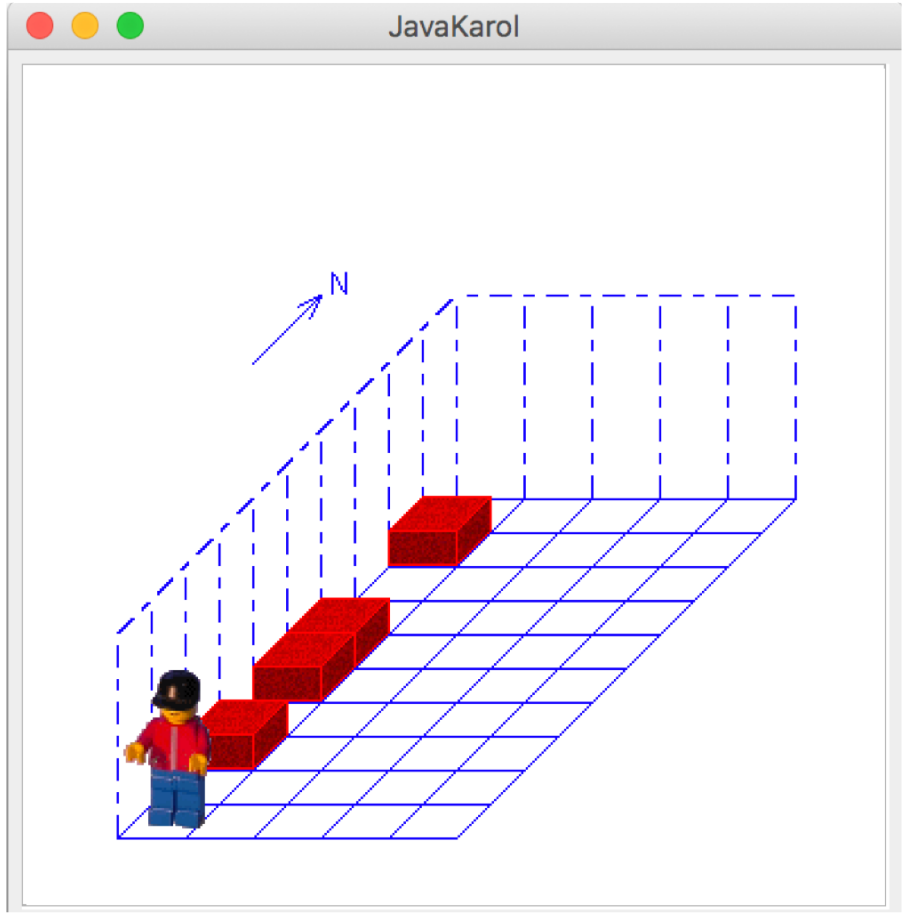


Tipp:
Um zu erfahren, welche Methoden die Klasse ROBOTER hat und erbt, kann man im Editor nach dem Eintippen von "karol." ctrl(strg)+Leertaste drücken und BlueJ listet alle Methoden auf.
Noch informativer ist es, die Projektdokumentation zu öffnen.
(BlueJ Menü -Tools - Project Documentation)

Übung 1 – JAVA Karol

e)

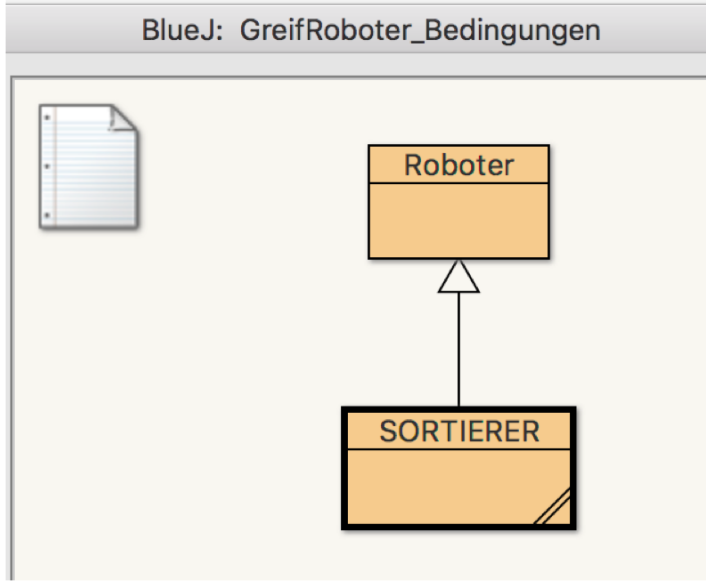
Schreibe die Methode *vorsichtigerSchritt()*, welche *karol* einen Schritt machen lässt, wenn keine Wand vor ihm ist.



Übung 2 – Greifroboter

a)

Kopiere das BlueJ-Projekt „GreifRoboter“ und ändere den Namen in „GreifRoboterBedingungen “ ab.

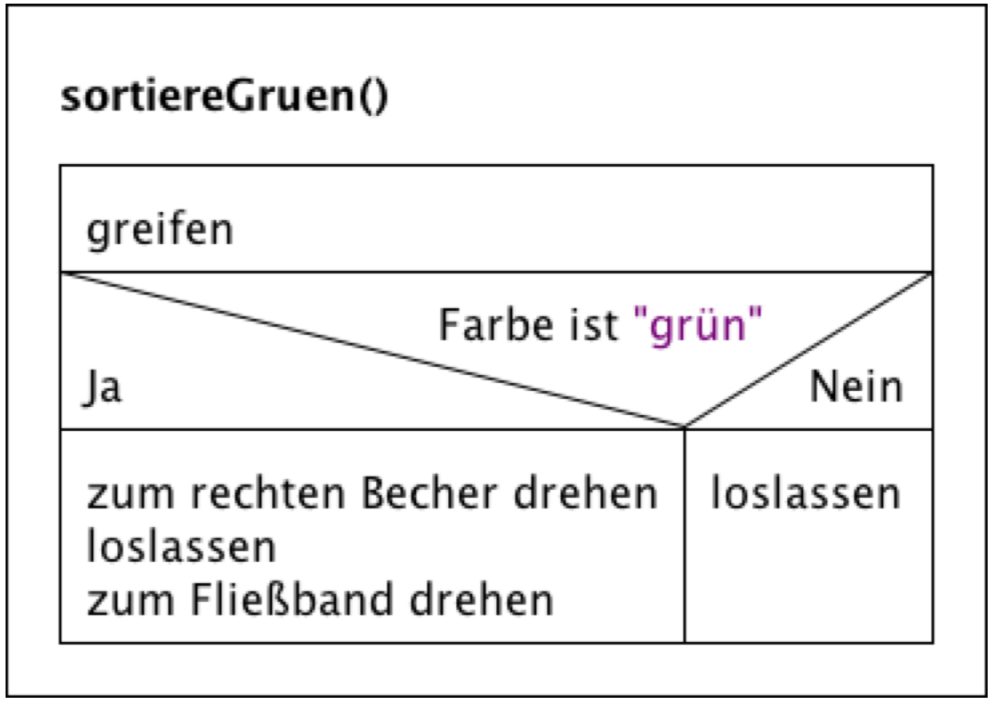


Erstelle eine Klasse SORTIERER,
die von Roboter erbt.
Denke an einen Konstruktor, der den Konstruktor der Oberklasse aufruft.

Übung 2 – Greifroboter

b)
Schreibe in der Klasse SORTIERER eine Methode *sortiereGruen(),*
welche die nächste Kugel greift und fallen lässt, wenn sie nicht grün ist.
Grüne Kugeln werden in den rechten Becher gelegt und der Arm wird anschließend wieder zum Fließband zurück bewegt.
Ob die Farbe grün ist, testest du so:

if(kugelfarbeGeben() == ”grün“) ...



Übung 2 – Greifroboter

c)
Schreibe eine weitere Methode *sortiere(String farbe)* , welche sich genauso verhält, aber die gewünschte Farbe als Übergabeparameter hat.

Erlaubte Farben sind:
“rot“
“magenta“
“grün“
“blau“
“weiß“

Übung 3 – Greifroboter

a)
Öffne das Projekt „GreifRoboterBedingungen“ aus Übung 2.

Schreibe eine Methode *nimmDrei(),* welche eine Kugel vom Fließband nimmt und in den **linken** Becher wirft, wenn die Nummer **nicht 3** ist.

Wenn die Nummer **3** ist, wird die Kugel in den **rechten** Becher geworfen.

Anschließend wird in beiden Fällen der Arm wieder zum Fließband zurück bewegt.

Tipp: Falls dir das Zurückbewegen des Arms auf diese Weise nicht gelingt, bewege den Arm in jedem einzelnen Fall gleich wieder zurück.

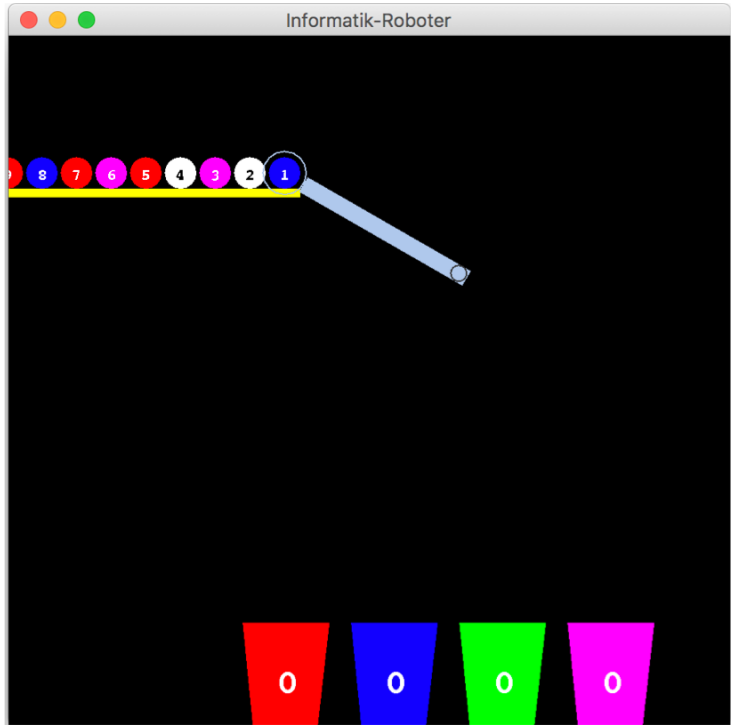
Übung 3 – Greifroboter

b)
Schreibe eine Methode *nimm(int nummer),* welche sich genauso verhält, aber die gewünschte Nummer als Übergabeparameter hat.

c)
Schreibe eine weitere Methode *nimmKleiner(int nummer),* welche alle Kugeln, deren Nummer kleiner als der Wert des Übergabeparameters sind, in den linken Becher wirft. Ist die Nummer gleich oder größer, wird sie in den rechten Becher geworfen.

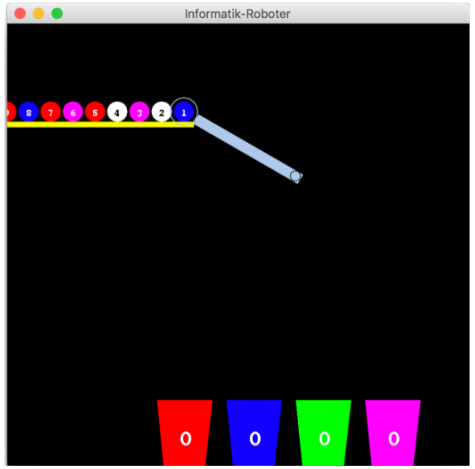
Übung 4 – Greifroboter2

a)
Öffne das Projekt „GreifRoboter2_Bedingungen“.
Erzeuge ein Objekt von SORTIERER.
Es gibt nun vier Becher.
Um den Greifarm zu den Bechern zu bewegen, sind jeweils, ausgehend von der Position am Fließband, die Winkel 60, 100, 135 und 170 nötig.
Der Winkel 210 (= -150) dreht den Arm nach rechts, so dass eine Kugel ins Leere fallen würde.



Teste diese Methoden.

Übung 4 – Greifroboter2



b)
Schreibe eine Methode *sortiereFarben(),*
die folgende Mehrfachauswahl mit Alternative
umsetzt.
Verwende *if - else if - else.*

sortiereFarben()

greifen				
Farbe der Kugel				
rot	blau	grün	magenta	sonst
zum 1. Becher drehen loslassen	zum 2. Becher drehen loslassen	zum 3. Becher drehen loslassen	zum 4. Becher drehen loslassen	waagrecht drehen loslassen
zum Fließband zurück drehen				

Übung 4 – Greifroboter2

c)*
Anstelle der Konstruktion
mit *if, else if* und *else* gibt es in Java auch die
***switch – case* – Konstruktion.**

Die Variable kann vom Typ char, int
oder ein String-Objekt sein.
(In unserem Beispiel die Farbe der Kugel)

Fall 1, Fall 2, ... steht dann jeweils einfach
für einen möglichen Wert der Variable.
(In unserem Beispiel “rot“, “blau“, ...)
In den Block mit “default“ kommt man, wenn
keiner der Fälle zutrifft.

Schreibe eine Methode *sortiereFarben2(),*
die dasselbe leistet wie *sortiereFarben().*
Verwende switch - case.

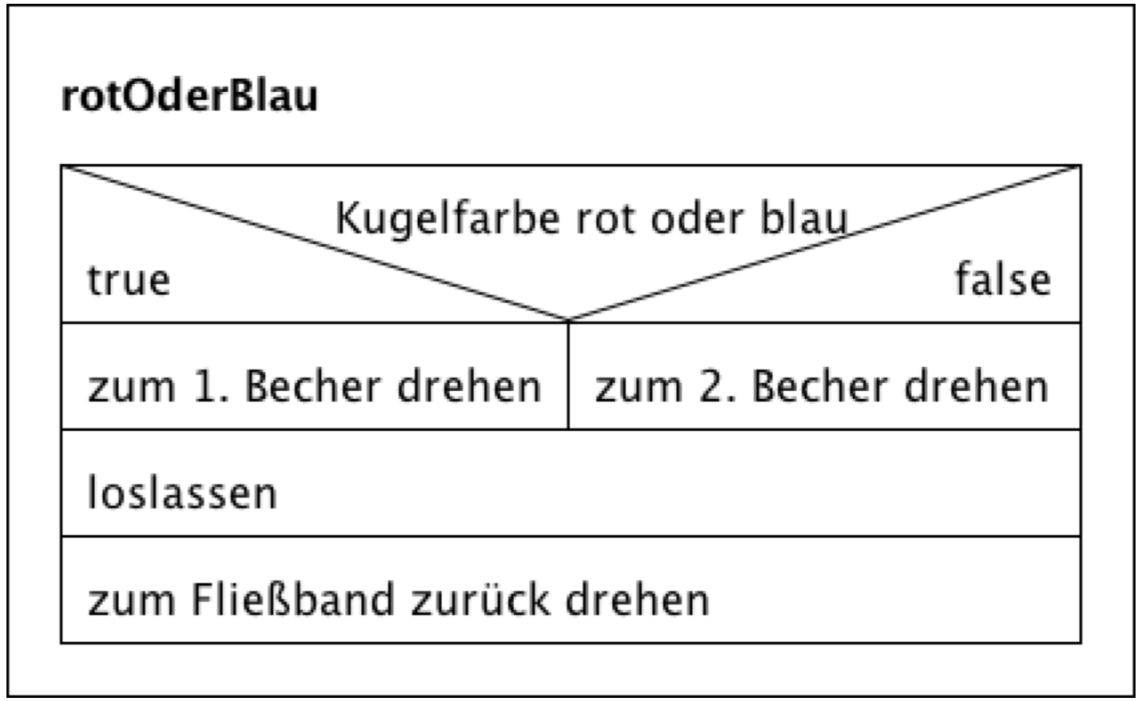
```
switch (Variable){
   case Fall 1:
      Anweisung 1;
      break;
   case Fall 2:
      Anweisung 2;
      break;
   case Fall 3:
      Anweisung 3;
      break;
   ...
   default :
      Anweisung 1;
      break;

}
```

Bedingte Anweisungen – boolesche Variablen

Oft muss man die booleschen Variablen in den Bedingungen verknüpfen.

Beispiel 1:



Bedingte Anweisungen – boolesche Variablen

Oft muss man die booleschen Variablen in den Bedingungen verknüpfen.

Beispiel 2:

rotUndGerade

Farbe rot und Nummer gerade	
true	false
zum 1. Becher drehen	zum 2. Becher drehen
loslassen	
zum Fließband zurück drehen	

Die Farbe muss rot sein und zugleich muss die Nummer gerade sein.

Bedingte Anweisungen – boolesche Variablen

Oft muss man die booleschen Variablen in den Bedingungen verknüpfen.

Beispiel 3:

nichtRotOderGerade

nicht (Farbe rot oder Nummer gerade)	
true	false
zum 1. Becher drehen	zum 2. Becher drehen
loslassen	
zum Fließband zurück drehen	

Beispiele hierfür:

Nr. 2, rot → 2.Becher

Nr.6, blau → 2.Becher

Nr.5, rot → 2.Becher

Nr.7, weiß →1.Becher

MERKE

Die OR (ODER) – Verknüpfung (in Java: a||b)

a	b	a OR b
false	false	false
false	true	true
true	false	true
true	true	true

a OR b liefert nur dann false, wenn sowohl a als auch b den Wert false hat.

MERKE

Die AND (UND) – Verknüpfung (in Java: a&&b)

a	b	a AND b
false	false	false
false	true	false
true	false	false
true	true	true

a AND b liefert nur dann true, wenn sowohl a als auch b den Wert true hat.

MERKE

Der NOT (NICHT) – Operator (in Java: !a)

a	!a
false	true
true	true

a ungleich b schreibt man in Java so:
a!=b oder !(a==b)

MERKE

! (a || b) = !a && !b

! (a && b) = !a || !b

Vergleiche dazu das Beispiel 3:
NICHT (rot ODER gerade) ist dasselbe wie NICHT rot UND NICHT gerade

Übung 5 – Greifroboter, logische Verknüpfungen

a)
Schreibe im Projekt GreifroboterBedingungen die Methoden
rotOderBlau()
rotUndGerade() und
nichtRotOderGerade()

Ob die Kugelnummer gerade ist, prüfst du so:
kugelnummergeben()%2==0
(Erläuterung: a%2 berechnet in Java den Rest bei der Division durch 2.)

b)*
Schreibe eine Methode **entwederRotOderGerade()**, die die Kugel in den 1. Becher ablegt, wenn die Kugel entweder rot oder die Nummer gerade ist.

Übung 6 – Animierter Ball 1

Öffne das Projekt Ball_Animation_1.
Die Klasse **BALL** erbt von KREIS, der Konstruktor erzeugt
ein Objekt gemäß der abgebildeten Objektkarte.

bALL1 : BALL	
int deltaX	-10
int deltaY	-5
private String farbe	"gelb"
private boolean sichtbar	true
private int radius	15
private int M_x	50
private int M_y	20

a)
Schreibe eine Methode **bewegen()** , welche die geerbte Methode
verschiebenUm(..., ...) aufruft.
Als Parameter für diese Methode wählst du die Attributwerte von **deltaX** und **deltaY**.
Hierdurch wird der Ball horizontal um den Wert **deltaX** und vertikal um den Wert **deltaY** verschoben.

Übung 6 – Animierter Ball 1

b) Ergänze die Methode **bewegen()** so, dass der Ball an den Rändern reflektiert wird.

Teil 1:

Das Spielfeld ist 800 Pixel breit und 600 Pixel hoch. Überlege dir zunächst Bedingungen, wann der Ball einen Bildschirmrand erreicht hat. Betrachte hierzu die Werte der geerbten Attribute M_x und M_y, das sind die Koordinaten des Mittelpunkts des Balls.

Daraus ergibt sich folgende Struktur der Fallunterscheidung:

```
wenn (linker Rand erreicht)
    dann ...
sonst wenn (rechter Rand erreicht)
    dann ...
sonst wenn (oberer Rand erreicht)
    dann ...
sonst wenn (unterer Rand erreicht)
    dann ...
```

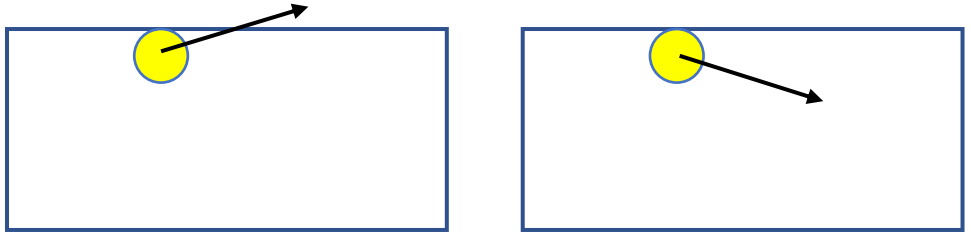
Übersetze diese Fallunterscheidung in JAVA. Die dann-Anweisung für die eigentlichen Reflexionen überlegst du dir im nächsten Schritt. Wenn du die Bedingungen erst testen möchtest, kannst du z.B. beim Erreichen eines Rands die Farbe des Balls ändern. Um eine Wand zu erreichen, kannst du die geerbte Methode setzeMittelpunkt(...,...) gut verwenden.

Übung 6 – Animierter Ball 1

b)
Ergänze die Methode **bewegen()** so, dass der Ball an den Rändern reflektiert wird.

Teil 2:
Schließlich muss noch jeweils die Reflexion am Rand umgesetzt werden.

Überlege dir dazu anhand einer Zeichnung, wie sich die Werte von deltaX und deltaY nach der Reflexion ändern müssen.



Hinweis:
beim Testen der Methode wirst du feststellen, dass der untere Rand nicht exakt bei 600, sondern bei 628 ist. Lass es vorerst so und rechne trotzdem mit 600, der Ball wird einfach etwas oberhalb des unteren Randes reflektiert.

Vererbung 2 – Überschreiben von Methoden

Erbt eine Klasse Attribute und Methoden einer Superklasse, so werden die Methoden unverändert ausgeführt.

Manchmal möchte man aber das **Verhalten einer geerbten Methode verändern.**

Beispiel:

Erzeuge ein Objekt der Klasse **SPIEL**. Die Methode tick() dieser Klasse wird nach einem bestimmten Zeitintervall automatisch immer wieder ausgeführt. Sie schreibt die Wörter „tick“ und „tack“ in ein Fenster.

Diese Methode wollen wir für eine Animation des Balls nutzen. Dazu soll sie natürlich nicht „tick“ und „tack“ schreiben sondern stattdessen die Methode **bewegen()** aufrufen.

MERKE

Möchte man das Verhalten einer geerbten Methode nachträglich ändern, so muss man die geerbte Methode in der Subklasse überschreiben.

Das Überschreiben eröffnet man mit dem Statement **@Override**.
In der nächsten Zeile schreibt man den Kopf der geerbten Methode genau so wie in der Superklasse.
Anschließend folgt der neue Code der Methode, der dann für ein Objekt der Subklasse so ausgeführt wird.

Beispiel:

```
@Override
public void tick() {
  this.ball.bewegen();
}
```

Übung 7 – Animierter Ball 2

Öffne das BlueJ-Projekt Ball_Animation1_Lsg.

Schreibe eine Klasse **BILLARD**, welche von Spiel erbt.

Deklariere und initialisiere in der Klasse **BILLARD** ein Referenzattribut der Klasse BALL und nenne es ball.

Überschreibe in der Klasse BILLARD schließlich die geerbte Methode **tick()** und rufe in ihrem Rumpf die Methode **bewegen()** des BALL-Objekts auf.

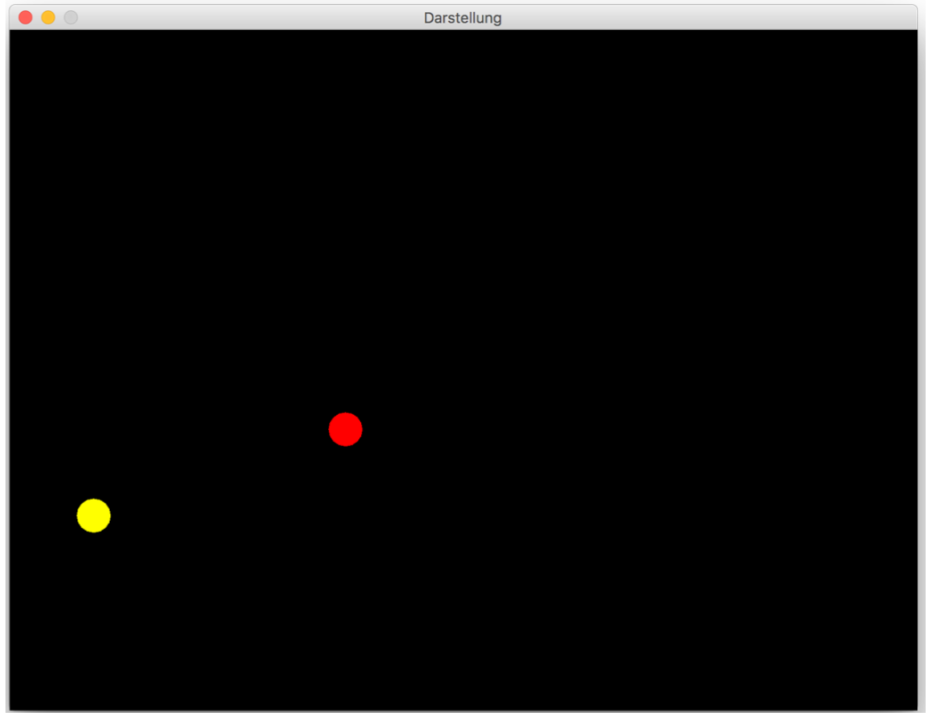
Teste die Methode, der Ball sollte sich nun automatisch bewegen und an den Rändern reflektiert werden.

Experimentiere mit der Methode **tickerIntervallSetzen(...)** und anderen Werten von **deltaX** und **deltaY,** so dass die Animation möglichst gleichmäßig ist.

Übung 8 – zwei Bälle

Kopieredas BlueJ-Projekt aus Übung7 und ändere den Namenin Ball_Animation_2_Baelle ab.

Ziel ist es, dass sich zwei Bälle im Spielfeld bewegen.



Übung 8 – zwei Bälle

Erstelle dazu in BALL einen zweiten Konstruktor mit folgenden **Übergabeparametern**

int dX, int dY, String farb, int mx, int my

Im Konstruktor erhalten die Attribute deltaX, deltaY, farbe, M_x und M_y die Werte der entsprechenden Übergabeparameter.

In der Klasse BILLARD nimmst du folgende Änderungen vor:

Es gibt zwei Attribute ball1 und ball2 der Klasse BALL.
Im Konstruktor erzeugst du diese Objekte mit Attributwerten deiner Wahl.

Übung 9* – zwei Bälle (für Spezialisten)

Die Bälle sollen auch untereinander reflektiert werden, wenn sie sich treffen.

Dies musst du in der Methode tick() erledigen.

Die Reflexion soll stattfinden, wenn der Abstand der x-Koordinaten der Mittelpunkte und der Abstand der y-Koordinaten der Mittelpunkte kleiner oder gleich der Summe der Radien ist.

Deklariere im Kopf der Klasse die Attribute dmx und dmy für die Abstände.
In der methode tick() werden sie berechnet und dadurch immer aktualisiert.

Schreibe die Bedingung für die Reflexion und überlege, wie sich die Bälle danach bewegen sollen.