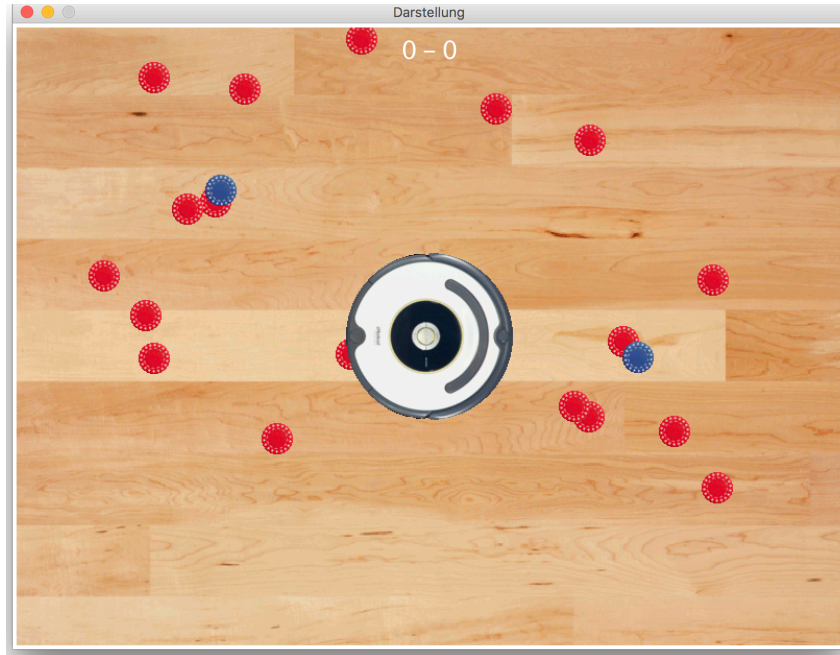


Projekt: Staubsauger-Roboter



Ziel:

Programmiere einen Staubsauger-Roboter, der von einer einfachen künstlichen Intelligenz (KI) gesteuert wird und dabei den Boden eines Raumes reinigt.

Hinweise:

Die Anleitung ist nicht mehr so detailliert wie im Projekt Ping-Pong. Viele Konzepte der engine alpha werden aber auch hier wieder angewendet, du kannst also deine Erfahrungen aus dem letzten Projekt gut nutzen.

Weitere Hilfen sind:

- **Das Handbuch für die engine alpha und eine Übersicht der Farben und Tasten-Codes** findest du offline im Ordner Skripten_und_Programme oder online unter http://engine-alpha.org/wiki/Download#Handbuch_2.0
- **Die Dokumentation für alle Klassen der engine alpha:** <https://docs.engine-alpha.org/v3.2.0/>
- **Die Projektdokumentation** findest du in BlueJ unter Tools – Project Documentation. Wenn du sie aktualisierst, werden auch deine eigenen Java-Doc-Kommentare aus dem Quelltext dort angezeigt. Eine Anleitung findest du im Ordner zum Ping-Pong-Projekt. Im Editor zeigst du mit strg(command)+Leertaste auf die verfügbaren Methoden und diese Kommentare zu.
- **Deine Mitschülerinnen und Schüler und selbstverständlich dein Informatiklehrer**

Für Fortgeschrittene:

- **Die offizielle Java-Dokumentation aller Klassen** <https://docs.oracle.com/javase/7/docs/api/>
- **Handbuch „Java ist auch eine Insel“** <http://openbook.rheinwerk-verlag.de/javainsel/index.html>

Kommentieren des Quelltextes:

Zur Erinnerung:

```
/**
```

*So wird ein Java-doc-Kommentar erzeugt, der dann in der Projektdokumentation zu sehen ist.

```
**/
```

Kommentiere darüber hinaus auch wichtige Schritte

innerhalb der Methoden:

```
// einzeliger Kommentar
```

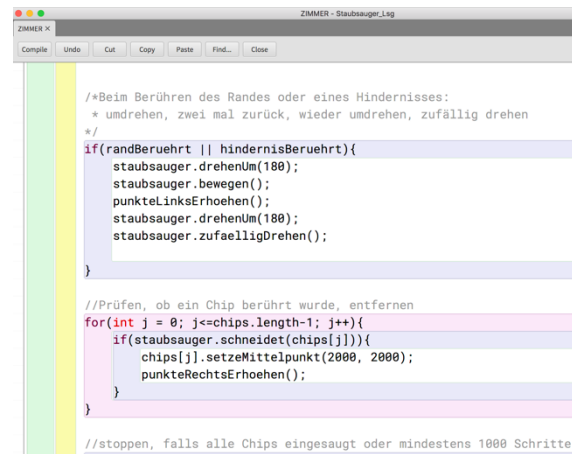
```
/*
```

mehrzeiliger

Kommentar

```
*/
```

Dies ist für die Lesbarkeit sehr wichtig und hilft dir außerdem, Programmierfehler leichter zu finden.



Vorbereitung:

Kopiere das **BlueJ-Projekt „Staubsauger_Vorlage“** in deinen Ordner.

Als Vorlage dienen dir drei Klassen:

- **RECHTECK**

Dient zur Darstellung der Wände des Raums.

- **BILD**

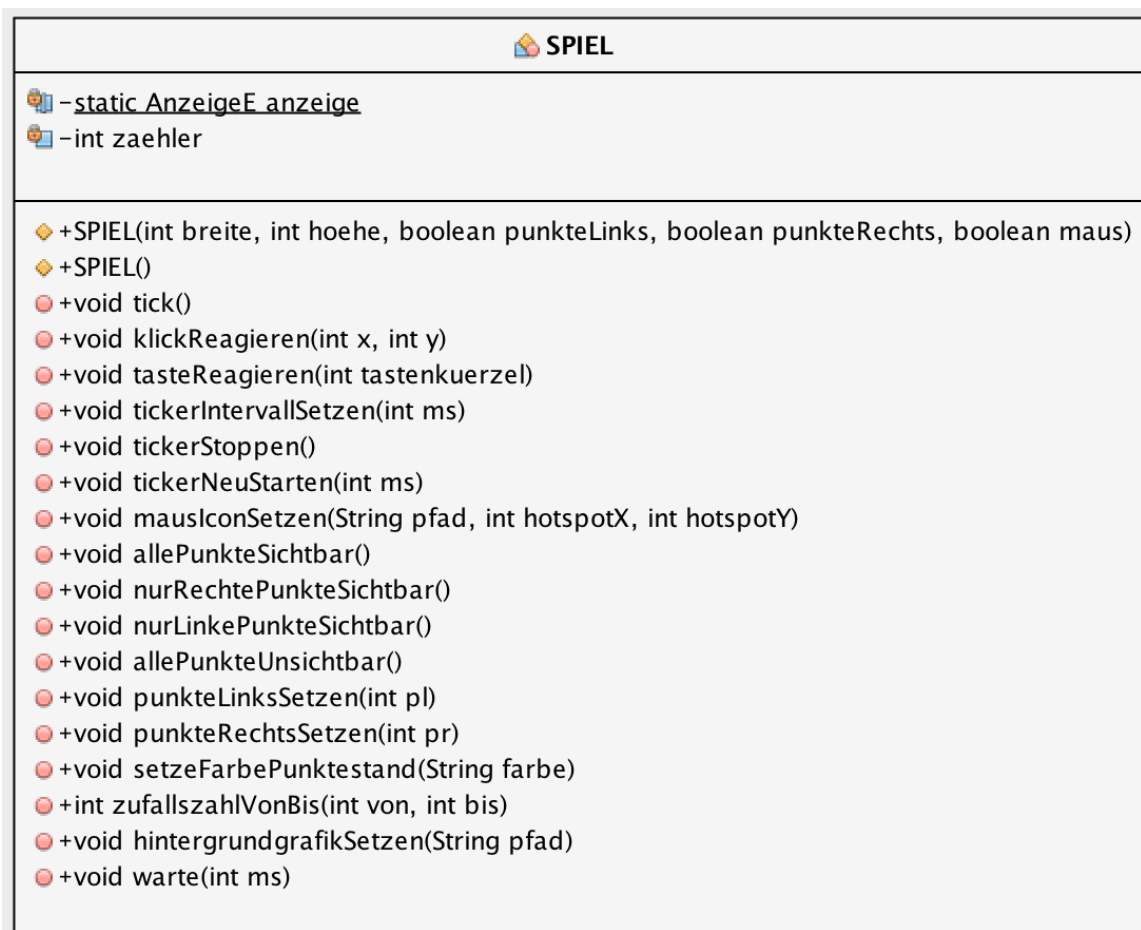
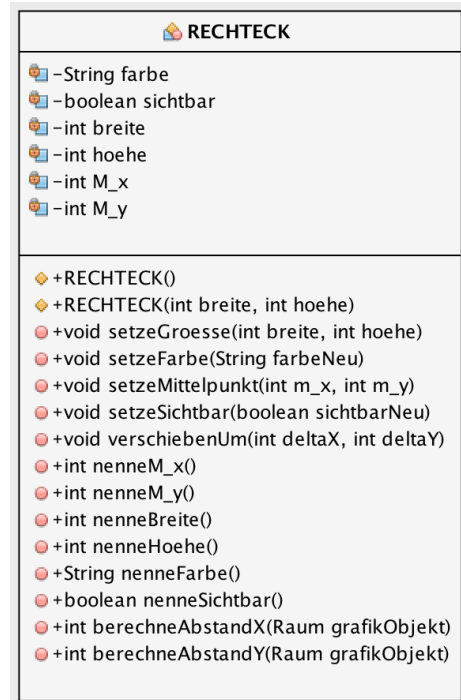
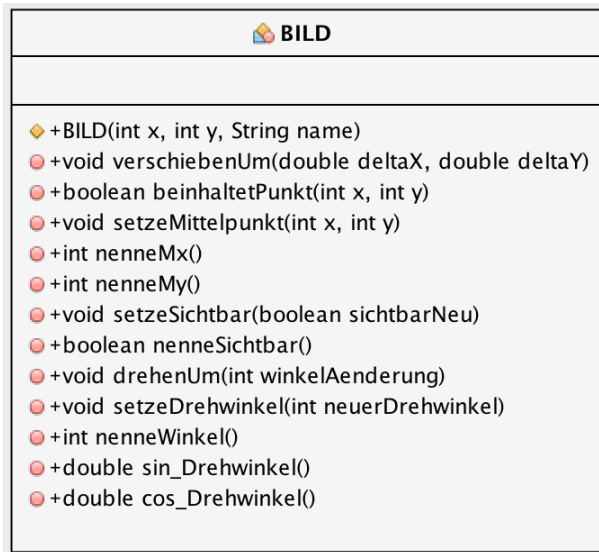
Mit dieser Klasse lassen sich Grafik-Dateien darstellen, wie zum Beispiel der Staubsauger oder die einzusammelnden Chips. Die Klasse bietet außerdem einige nützliche Methoden, die bei der Steuerung des Roboters helfen.

- **SPIEL**

Wie schon im Ping-Pong-Projekt dient diese Klasse dazu, Bewegung in das Szenario zu bringen. Es gibt insbesondere die Methode tick und tasteReagieren, die automatisch immer wieder aufgerufen werden und überschrieben werden können.

In den folgenden Schritten wirst du die Klassen **CHIP**, **HINDERNIS**, **STAUBSAUGER** und **ZIMMER** programmieren.

Klassenkarten:



Schritt 1: Erkundung der Klasse BILD

Betrachte den Konstruktor:

```
/**
 * BILD Konstruktor
 *
 * @param x      x-Koordinate im Fenster (Pixel)
 * @param y      y-Koordinate im Fenster (Pixel)
 * @param name   Name der Grafik-Datei (im Projekt-Ordner)
 */
public BILD(int x, int y, String name) {
    super(x, y, name);
    this.setzeMittelpunkt(x, y);
}
```

name ist der vollständige Name der Grafik-Datei einschließlich der Endung. Die Grafik-Datei muss im BlueJ-Projekt-Ordner liegen.

1a) Erzeuge der Reihe nach je ein BILD-Objekt mit der Staubsauger ("Robot.gif") - und der Chip-Grafik.

1b) Experimentiere bei diesen Objekten mit den Methoden **verschiebenUm(...)**, **drehenUm(...)**, **nenneWinkel()**, **setzeDrehwinkel(...)**

Achte insbesondere auf die Orientierung der Winkel.



Schritt 2: Die Klassen CHIP und HINDERNIS

2a)

Erstelle die Klassen CHIP und HINDERNIS, welche von BILD erben.

Der Konstruktor von CHIP hat zwei Übergabeparameter, mit denen die Koordinaten des Mittelpunkts festgelegt werden können.

Im Konstruktor selbst wird der geerbte Konstruktor der Klasse BILD aufgerufen.

Dieser hat drei Übergabeparameter. Übernimm die Koordinaten des Mittelpunkts und wähle als dritten Parameter "Chip.gif".

2b)

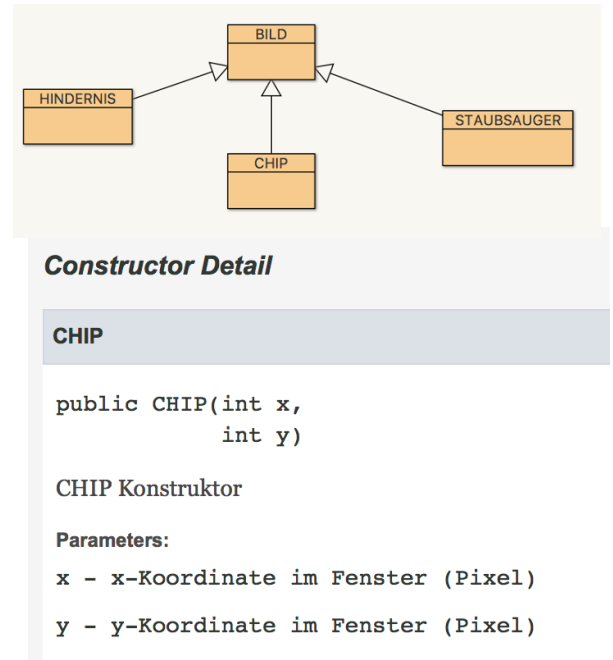
Verfahre analog mit der Klasse HINDERNIS.

Hier wählst du "ChipBlau.gif" als Grafik-Datei.

Mehr ist hier nicht zu tun, alle benötigten Methoden werden geerbt.

2c)

Teste deine Klassen, indem du einige CHIP- und HINDERNIS-Objekte an verschiedenen Orten erzeugst.



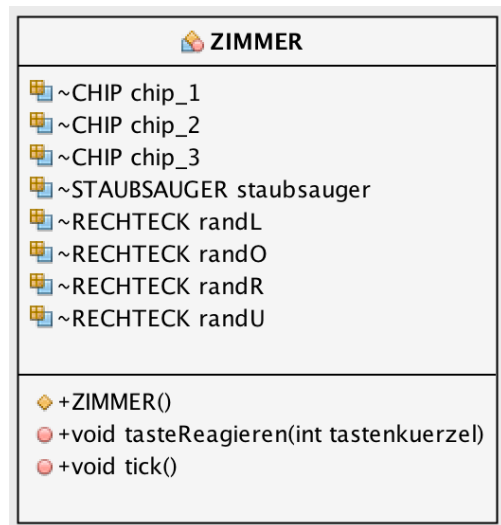
Schritt 3: Die Klasse ZIMMER – Teil I

3a)

Erstelle eine Klasse ZIMMER, die von SPIEL erbt.

Beginne mit dem Konstruktor:

Der Konstruktor hat keine Übergabeparameter und ruft den Konstruktor der Klasse SPIEL auf, welcher fünf Übergabeparameter hat.



SPIEL

```
public SPIEL(int breite,
            int hoehe,
            boolean punkteLinks,
            boolean punkteRechts,
            boolean maus)
```

Erstellt ein Spiel. Startet die Anzeige.

Parameters:

punkteLinks - ist dieser Wert true, so sieht man links eine Punkteanzeige. Ist er false sieht man keine.

punkteRechts - ist dieser Wert true, so sieht man rechts eine Punkteanzeige. Ist er false sieht man keine.

maus - ist dieser Wert true, wird eine Maus im Spiel angezeigt und verwendet. Ist er false, gibt es keine Maus.

Erzeuge auf diese Weise ein Fenster von 800px Breite, 600px Höhe, einem linken und einem rechten Punktestand und ohne Mauszeiger.

3b)

Von der Klasse SPIEL erbst du die Methode **hintergrundgrafikSetzen(...)**.

Setze gleich nach dem Aufruf des Super-Konstruktors die Datei "Boden.gif" als Hintergrund.

3c)

Deklariere drei Referenzattribute vom Typ CHIP mit den Namen chip_1, chip_2 und chip_3 und initialisiere sie im Konstruktor. Die Koordinaten der Mittelpunkte sollen zufällig gesetzt werden. Nutze dazu die geerbte Methode **zufallszahlVonBis(..., ...)**.

Die Attribute staubsauger, randL, randO, randR und randU werden in Schritt 4 und 5 erstellt.

Anstelle dieser vorgefertigten Methode kannst du auch Java-Klassen zum Erstellen von Zufallszahlen verwenden:

Methode 1:

```
Math.random();
```

liefert eine rationale Zufallszahl vom Typ double aus dem Intervall [0;1[.

Die Methoden dieser Klasse haben die Bezeichnung static. Das bedeutet, dass man kein Objekt benötigt, um sie aufzurufen.

eine ganzzahlige Zufallszahl aus dem Intervall [0;5] kannst du dann z.B. mit der Anweisung

```
(int) (6*Math.random());
```

erzeugen.

(int) wandelt dabei die rationale in eine ganze Zahl um.

Methode 2:

Importiere dazu am Anfang der Klasse noch vor dem Klassenkopf das benötigte Java-Package:

```
import java.util.*;
```

Deklariere dann ein Objekt zgen der Klasse Random und initialisiere es im Konstruktor:

```
zgen = new Random();
```

Nun kannst du die Methoden der Klasse Random nutzen.

Beispiel:

```
zgen.nextInt(6)
```

erzeugt eine ganzzahlige Zufallszahl aus dem Intervall [0;5].











Schritt 4: Die Klasse STAUBSAUGER

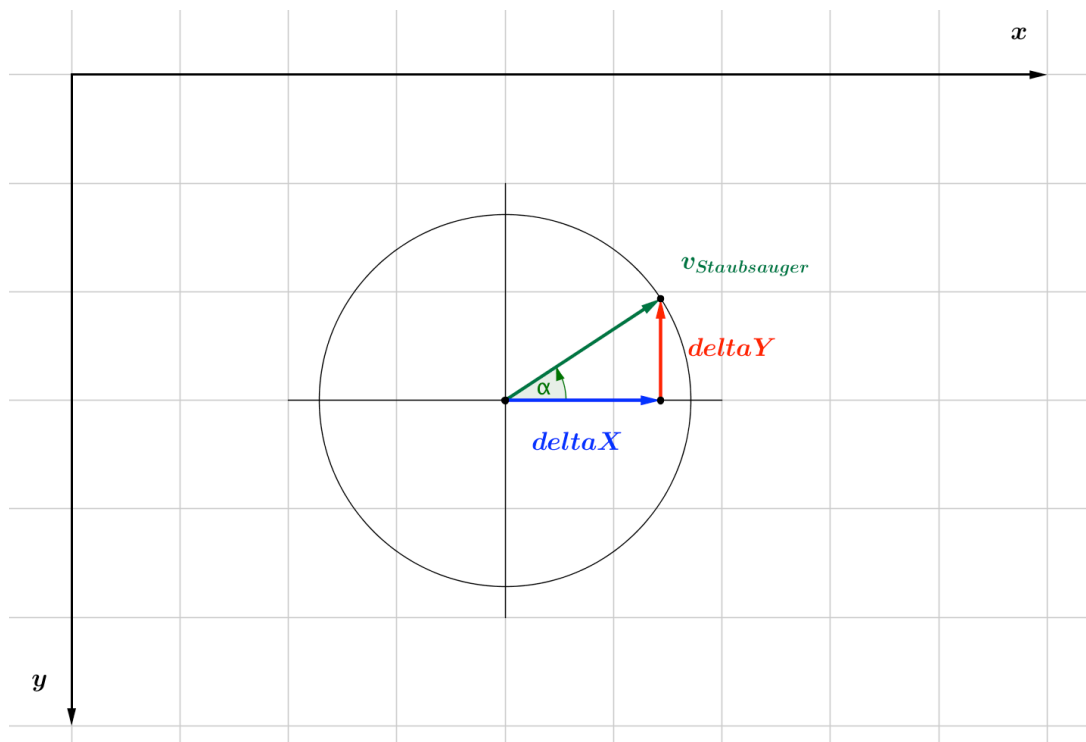
Vorüberlegung:

Der Staubsauger soll sich immer mit konstanter Geschwindigkeit fortbewegen, egal in welche Richtung er gerade blickt.

Δx und Δy geben an, wie weit sich der Staubsauger in x- und in y-Richtung bewegt.

Diese Werte müssen Abhängigkeit des Drehwinkels α berechnet werden:

STAUBSAUGER	
 -double Δx	
 -double Δy	
 -int geschwindigkeit	
<hr/>	
 +STAUBSAUGER(int M_x, int M_y)	
 +STAUBSAUGER()	
 -void berechneSchrittweitenNeu()	
 +void setzeDrehwinkel(int winkel)	
 +void drehenUm(int winkel)	
 +void bewegen()	
 +void setzeGeschwindigkeit(int vNeu)	



Für unser Koordinatensystem gilt also:

$$\Delta x = \text{geschwindigkeit} \cdot \cos(\alpha)$$

$$\Delta y = -\text{geschwindigkeit} \cdot \sin(\alpha)$$

Die Werte der Sinus- und der Cosinusfunktion berechnest du mit den geerbten Methoden ***sin_Drehwinkel()*** und ***cos_Drehwinkel()*** der Klasse BILD.

4a)

Erstelle eine Klasse STAUBSAUGER, die von BILD erbt.

Der erste Konstruktor soll die Mittelpunktskordinaten als Übergabeparameter haben. Diese werden an den Konstruktor der Superklasse weitergegeben.

Initialisiere die Werte für *geschwindigkeit*, *deltaX* und *deltaY*:

geschwindigkeit=10

deltaX=10

deltaY=0

Der zweite Konstruktor ohne Übergabeparameter soll den ersten mit den Werten 400 und 300 aufrufen.

Verwende dazu einfach die Anweisung ***this(400,300);***

4b)

Schreibe eine Methode ***berechneSchrittweitenNeu()***, welche z.B. nach einer Veränderung des Drehwinkels *deltaX* und *deltaY* neu berechnet.

Die Klasse STAUBSAUGER erbt zwar von der Klasse BILD zwei Methoden zum Drehen der Grafik, aber diese drehen nur die Grafik und berechnen nach der Drehung die Schrittweiten nicht neu.

Um dies zu lösen, überschreibt man die Methode und lässt dort zuerst den geerbten Code ausführen und ergänzt anschließend geeignet. Die Technik heißt **Erweitern von Methoden**.

4c)

Überschreibe in der Klasse STAUBSAUGER die Methoden ***setzeDrehwinkel(...)*** und ***drehenUm(...)***.

Das oben beschriebene Erweitern von Methoden setzt man folgendermaßen um:

```
@Override
public void setzeDrehwinkel (int winkel){
    super.setzeDrehwinkel(winkel);
    this.berechneSchrittweitenNeu();
}
```

Schreibe analog die Methode ***drehenUm(...)***.

4d)

Schreibe die Methoden ***bewegen()*** und ***setzeGeschwindigkeit(int vNeu)***.

bewegen() verschiebt den Staubsauger um *deltaX* und *deltaY*.

setzeGeschwindigkeit(int vNeu) setzt den Wert von *geschwindigkeit* auf den Wert des Übergabeparameters *vNeu*. Denke beim Verändern der Geschwindigkeit daran, auch die Schrittweiten neu berechnen zu lassen.

Schritt 5: Die Klasse ZIMMER – Teil II

5a)

Deklariere in der Klasse ZIMMER vier Referenzattribute vom Typ `RECTECK` als Ränder.

Initialisiere diese Objekte so, dass sie jeweils am Rand des Grafik-Fensters liegen.

5b)

Deklariere außerdem ein Referenzattribut vom Typ `STAUBSAUGER` und initialisiere es im Konstruktor.

5c)

Überschreibe die geerbte Methode ***tasteReagieren(...)***.

Bei Betätigung der Pfeiltasten (links, rechts) soll sich der Staubsauger jeweils um 10° gegen bzw. im Uhrzeigersinn drehen.

Eine Datei mit Tastenbelegungen findest du offline im Ordner Skripten_und_Programme.

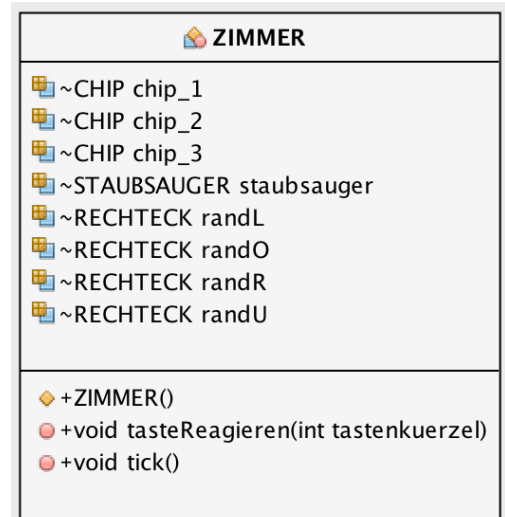
5d)

Überschreibe die geerbte Methode ***tick()***.

Bei jedem Aufruf soll sich der Staubsauger um einen Schritt bewegen.

5e)

Teste deine Klasse ZIMMER, indem du ein Objekt davon erzeugst. Versuche den Staubsauger-Roboter über das Grafik-Fenster zu steuern. Verhält sich alles wie erhofft?



Schritt 6: Chips einsammeln

Bisher reagiert der Staubsauger nicht auf die Chips.

6a)

Begib dich in die Methode ***tick()*** und dort unmittelbar unter die Zeile, welche den Staubsauger bewegt.

Füge dort für jeden Chip eine bedingte Anweisung ein, welche prüft, ob der Staubsauger einen Chip schneidet. Verwende dazu die geerbte Methode ***boolean schneidet(...)***.

Falls der Staubsauger einen Chip berührt, soll der Chip an den Ort (2000,2000) verschoben werden.

Auf diese Weise ist der Chip aus dem Fenster verschwunden und es sieht aus, als ob er eingesammelt worden wäre.










6b)

Erzeuge ein Objekt deiner Klasse ZIMMER und prüfe, ob der Staubsauger nun wirklich die Chips einsammelt.

Schritt 7: Mehr Chips durch Arrays

Wenn du mehr als nur einige wenige Chips verteilen möchtest, z.B. 20 Stück, dann müsstest du im Deklarations- und auch im Initialisierungsteil 20 Mal den selben Code schreiben, bis auf die Nummer des Chips. Ähnlich wäre es an der Stelle im Code, wo die Chips eingesammelt werden.

In allen drei Situationen kannst du den Code erheblich verkürzen, indem du alle Chips in einem Array vom Typ CHIP verwaltest.

ZIMMER
 ~CHIP[] chips  ~STAUBSAUGER staubsauger  ~RECHTECK randL  ~RECHTECK randO  ~RECHTECK randR  ~RECHTECK randU
 +ZIMMER()  +void tasteReagieren(int tastenkuerzel)  +void tick()

7a)

Lösche im Deklarationsteil die drei Chips und deklariere stattdessen ein Array vom Typ CHIP mit dem Namen chips.

7b)

Initialisiere im Konstruktor das Array mit der Länge 20.

Erzeuge mithilfe einer Zählschleife die 20 Chips mit zufällig gesetzten Koordinaten.

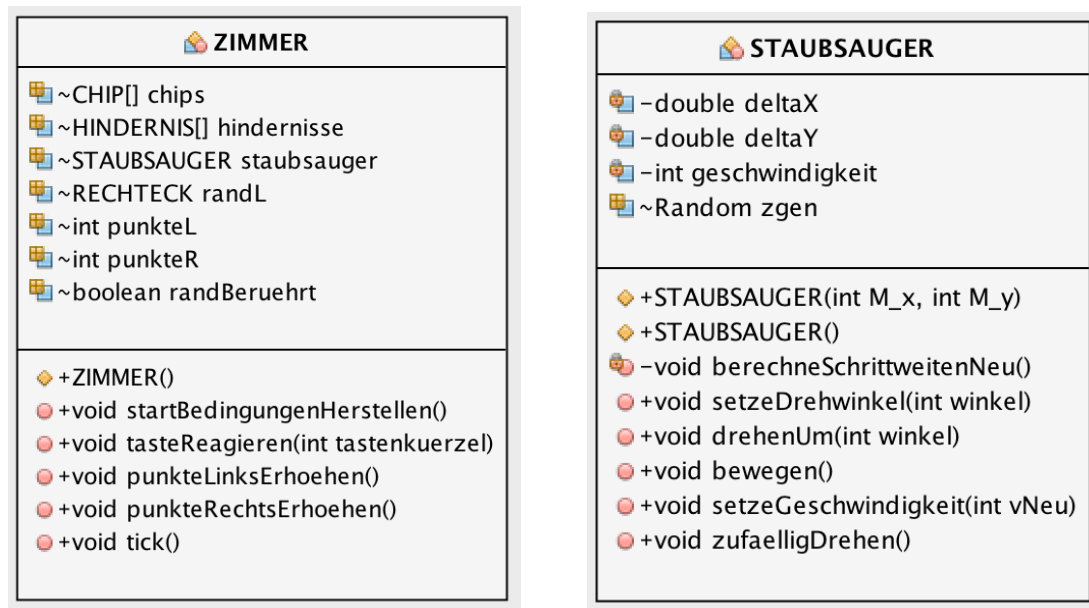
7c)

Begib dich in die Methode *tick()* und schreibe auch dort eine Zählschleife, in der du prüfst, ob einer der Chips vom Roboter geschnitten wird.

7d)

Teste deine Veränderungen, indem du wieder ein Objekt der Klasse ZIMMER erzeugst.

Schritt 8: Künstliche Intelligenz



Der Staubsauger soll sich nun automatisch im Zimmer bewegen und die Chips einsammeln. Auf diese Weise wird die Steuerung über die Tastatur unnötig.

Berührt der Staubsauger einen Rand, soll er sich umdrehen, etwas zurückfahren und sich eine andere Richtung suchen.

Eine mögliche Umsetzung ist es, in der Klasse STAUBSAUGER eine Methode **zufaelligDrehen()** zu schreiben, die den Staubsauger um einen zufälligen Winkel, z.B. zwischen 30° und 90° dreht. Dazu benötigst du die Java-Klasse Random. Eine kurze Beschreibung findest im Exkurs nach Schritt 3.

Du kannst dir auch andere Strategien des Staubsauger-Roboters überlegen.

Nutze auch die von SPIEL geerbten Methoden, um Punktestände anzuzeigen. Ändere z.B. den linken Punktestand bei jedem Schritt und den rechten Punktestand bei jedem eingesammelten Chip.

Stoppe den Ticker, wenn der letzte Chip eingesammelt wurde. Bei der Taste N sollen die Chips zufällig neu verteilt und der Ticker neu gestartet werden.

Du kannst auch die bisher noch nicht genutzte Bilddatei *ChipBlau.gif* verwenden, um unüberwindliche Hindernisse im Raum zu verteilen.