Projekt: Staubsauger-Roboter

Immer beliebter werden die kleinen automatischen Haushaltshilfen.

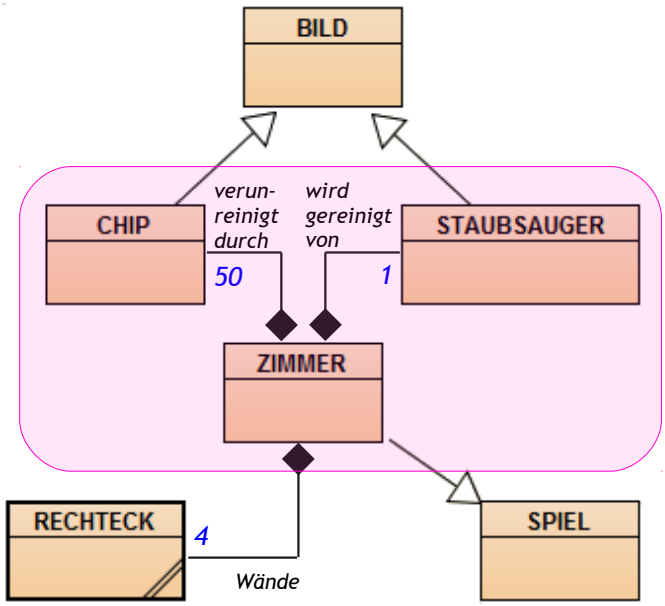
Du sollst nun einen Staubsauger-Roboter programmieren, der – gesteuert von einer künstlichen Intelligenz (KI) – den Boden eines Raums reinigt.

Als Vorlage dienen dir drei Klassen:

- **RECHTECK**
  Damit kannst du die Wände des Raums darstellen.

- **BILD**
  Damit lassen sich Grafik-Dateien in unserem Fenster darstellen. So kannst du den Staubsauger und auch die „Chips“, welche eingesammelt werden sollen, darstellen. Diese Klasse bietet außerdem einige nützliche Methoden, welche uns gute Dienste bei der Steuerung des Roboters leisten werden.

- **SPIEL**
  Diese Klasse dient wieder dazu, unserem Szenario Leben einzuhauchen durch automatisch immer wieder kehrende Methoden-Aufrufe und Reaktion auf Tastatur-Ereignisse.
  Auch diese Klasse bringt einige weitere nützliche Methoden mit, welche du brauchen wirst.



Du wirst nach diesen Vorgaben die Klassen **CHIP**, **STAUBSAUGER** und **ZIMMER** programmieren.

Betrachte zunächst die Klassen-Karten der gegebenen Klassen:

BILD

+ BILD(x : int, y : int, name : String)
+ beinhaltetPunkt(x : int, y : int) : boolean
+ cos_Drehwinkel() : float
+ drehenUm(winkelAenderung : int) : void
+ nenneMx() : int
+ nenneMy() : int
+ nenneSichtbar() : boolean
+ nenneWinkel() : int
+ schneidet(r : Raum) : boolean
+ setzeDrehwinkel(neuerDrehwinkel : int) : void
+ setzeMittelpunkt(x : int, y : int) : void
+ setzeSichtbar(sichtbarNeu : boolean) : void
+ sin_Drehwinkel() : float
+ verschiebenUm(deltaX : float , deltaY : float) : void

RECHTECK

M_x : int
M_y : int
breite : int
farbe : String
hoehe : int
sichtbar : boolean

+ RECHTECK()
+ RECHTECK(breite : int, hoehe : int)
+ berechneAbstandX(grafikObjekt : Raum) : int
+ berechneAbstandY(grafikObjekt : Raum) : int
+ nenneBreite() : int
+ nenneFarbe() : String
+ nenneHoehe() : int
+ nenneM_x() : int
+ nenneM_y() : int
+ nenneSichtbar() : boolean
+ schneidet(grafikObjekt : Raum) : boolean
+ setzeFarbe(farbeNeu : String) : void
+ setzeMittelpunkt(m_x : int, m_y : int) : void
+ setzeSichtbar(sichtbarNeu : boolean) : void
+ verschiebenUm(deltaX : int, deltaY : int) : void

SPIEL

anzeige : AnzeigeE
zaehler : int

+ SPIEL(breite : int, hoehe : int, punkteLinks : boolean, punkteRechts : boolean, maus : boolean)
+ SPIEL()
+ allePunkteSichtbar() : void
+ allePunkteUnsichtbar() : void
+ hintergrundgrafikSetzen(pfad : String) : void
+ klickReagieren(x : int, y : int) : void
+ mausIconSetzen(pfad : String, hotspotX : int, hotspotY : int) : void
+ nurLinkePunkteSichtbar() : void
+ nurRechtePunkteSichtbar() : void
+ punkteLinksSetzen(pl : int) : void
+ punkteRechtsSetzen(pr : int) : void
+ tasteReagieren(tastenkuerzel : int) : void
+ tick() : void
+ tickerIntervallSetzen(ms : int) : void
+ tickerNeuStarten(ms : int) : void
+ tickerStoppen() : void
+ zufallszahlVonBis(von : int, bis : int) : int

Schritt 1: Erkundung der Klasse BILD

1 a) Sieh dir die Klassen-Karte der Klasse BILD genau an, damit du einen ersten Eindruck bekommst, welche Methoden du erbst, wenn du später deine Klasse CHIP und STAUBSAUGER von dieser Klasse ableitest.

1 b) Betrachte insbesondere den Konstruktor:
x und ***y*** geben dem Mittelpunkt der Grafik in Pixel innerhalb des Grafik-Fensters an.
name ist der vollständige Name der Grafik-Datei (incl. Datei-Endung).
Die Grafik-Datei muss im BlueJ-Projekt-Ordner liegen.

Erzeuge nun der Reihe nach je ein BILD-Objekt mit der Staubsauger-Grafik und der Chip-Grafik.

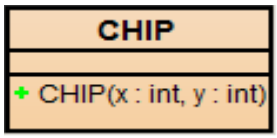
1 c) Experimentiere bei diesen Objekten etwas herum mit den Methoden
`verschiebenUm(...)`, `drehenUm(...)`, `nenneWinkel()`, `setzeDrehwinkel(...)`, …

Der Staubsauger im Bild blickt nach rechts.
Welchem Wert für den Drehwinkel entspricht das? _____

Dreht sich der Staubsauger im oder gegen den Uhrzeigersinn, wenn man der Methode ***drehenUm(...)*** *einen* ***positiven Wert*** *übergibt?* _____

Schritt 2: Klasse CHIP

2) Erstelle eine Klasse CHIP, welche von BILD erbt.



Der Konstruktor soll zwei Übergabe-Parameter haben, mit denen die Koordinaten des Mittelpunkts festgelegt werden können.
Vergiss nicht, den geerbten Super-Konstruktor aufzurufen. Aber ***Vorsicht****, dieser hat drei Übergabe-Parameter.*
Reiche die Parameter für die Mittelpunkts-Koordinaten einfach durch und gib als dritten Parameter den konkreten Wert "Chip.gif" an.

Mehr ist hier nicht zu tun, da du alle benötigten Methoden bereits erbst.

Teste deine Klasse, indem du drei verschiedene CHIP-Objekte an verschiedenen Orten erzeugst.

Schritt 3: Klasse ZIMMER – Teil I

3 a) Erstelle eine Klasse ZIMMER, welche von SPIEL erbt.
Rufe im Rumpf des Konstruktors den Super-Konstruktor mit den Übergabe-Parametern auf. Erstelle auf diese Weise ein Fenster von 800 Pixel Breite, 600 Pixel Höhe, einem linken und rechten Punktestand aber ohne Mauszeiger.

3 b) Du erbst von der Klasse SPIEL die Methode `hintergrundgrafikSetzen(...)`.
Als Übergabe-Parameter wird der Dateiname des Bildes übergeben, das als Hintergrund dienen soll. Die Bilddatei muss im Projektordner liegen.
Setze gleich nach dem Aufruf des Super-Konstruktors die Datei `Boden.gif` *als Hintergrund.*

3 c) *Deklariere drei Referenz-Attribute vom Typ CHIP und nenne sie chip_1, chip_2, chip_3.*
Vergiss nicht, die Chips im Konstruktor zu initialisieren.
Nutze für die Mittelpunkt-Koordinaten im Konstruktor der Chips die von SPIEL geerbte Methode zufallszahlVonBis(... , ...) , um die x- bzw- y-Koordinate jedes einzelnen Chips zufällig zu wählen.

3 d) Teste deine Klasse, indem du ein Objekt davon erstellst. Sieht du drei Chips?

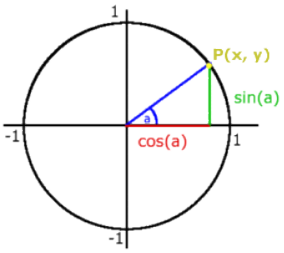
Schritt 4: *Klasse STAUBSAUGER*

STAUBSAUGER
− deltaX : double − deltaY : double − geschwindigkeit : int
+ STAUBSAUGER() + STAUBSAUGER(M_x : int, M_y : int) + bewegen() : void + drehenUm(winkelAenderung : int) : void + setzeDrehwinkel(neuerDrehWinkel : int) : void + setzeGeschwindigkeit(v : int) : void − berechneSchrittweiten() : void

Bevor du beginnst, diese Klasse zu programmieren, musst du dich ein klein Wenig an bekannte mathematische Zusammenhänge erinnern:

Der Staubsauger soll sich immer mit konstanter Geschwindigkeit fortbewegen, egal in welche Richtung er gerade blickt. Für dieie Methode *verschiebenUm(... , ...)* brauchst du aber Werte für *deltaX und deltaY* *(Schrittweiten in Pixel)*.
Wie groß sind diese Werte aber bei einem Drehwinkel von 30°, 45°, 53°, … ?

Der Drehwinkel unseres Roboters wird dabei genau wie am Einheitskreis angegeben. Wenn der Roboter den Drehwinkel α hat, dann blickt er also längs der blauen Strecke in Richtung des Punktes P. Wenn er nun die *Geschwindigkeit 1* hat, macht er einen *Schritt längs der blauen Linie*, so beträgt der Wert für die Schrittweite *deltaX* *(rot)* nun gerade *cos(α)* und sein *deltaY* *(grün)* hat den Wert *sin(α)*.



Allgemein betragen die Werte für die Schrittweiten damit

this.deltaX = this.geschwindigkeit * cos_Drehwinkel() und
this.deltaY = this.geschwindigkeit * sin_Drehwinkel()

4 a) Erstelle eine Klasse STAUBSAUGER, welche von BILD erbt.
Deklariere die Attribute `deltaX`, `deltaY` und `geschwindigkeit` (s. Klassen-Karte).

Der erste Konstruktor soll die Mittelpunkts-Koordinaten als Übergabe-Parameter haben. Diese werden an den Super-Konstruktor übergeben und anschließend die Attribute wie in der Objekt-Karte ersichtlich initialisiert.

Der zweite Konstruktor ohne Übergabe-Parameter soll den ersten mit den Werten 400 und 300 aufrufen.
(s. Überladen von Methoden).

robot: STAUBSAUGER
geschwindigkeit = 10 deltaX = 10 deltaY = 0

4 b) Schreibe eine private Methode `berechneSchrittweitenNeu()`, welche (z.B. nach einer Veränderung des Drehwinkels) `deltaX` und `deltaY` neu berechnet.
Wende hierzu die oben geschilderten trigonometrischen Zusammenhänge an.

Die Klasse STAUBSAUGER erbt zwar von der Klasse BILD zwei Methoden zum Drehen der Grafik aber diese Drehen nur die Grafik, berechnen aber unsere Schrittweiten nicht neu, so dass der Staubsauger nach einer Drehung nicht in die neue Blickrichtung fahren würde.

Du sollst die Funktion dieser Methoden nun erweitern. Man nennt diese Technik auch tatsächlich ***Erweitern von Methoden*** da man erst den geerbten Code ausführen lässt und anschließend noch etwas ergänzt.

4 c) Überschreibe deshalb die Methoden `setzeDrehwinkel(...)` und `drehenUm(...)`. Wir wenden hier die Technik des erweiternden Überschreibens ein:

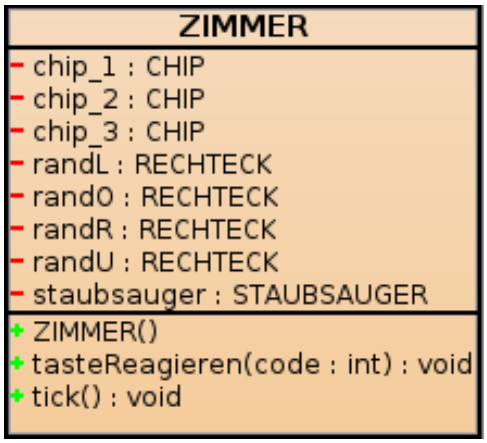
Zuerst lassen wir den Original-Code der Methode ausführen. Anschließend ergänzen wir den Methodenaufruf der gerade geschriebenen Methode.

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

analog: `drehenUm(...)`

4 d) Schreibe die Methoden `bewegen()` und `setzeGeschwindigkeit(int vNeu)`. Denke beim Verändern der Geschwindigkeit unbedingt daran, auch die Schrittweiten neu berechnen zu lassen ...

Schritt 5: Klasse ZIMMER – Teil II



5 a) Deklariere in deiner Klasse ZIMMER vier Referenz-Attribute vom Typ RECHTECK als Ränder. Initialisiere diese Objekte im Konstruktor so, dass sie jeweils am Rand des Grafik-Fensters liegen.
Mache dir hierfür vorher eine Zeichnung in der du Breite, Höhe und Mittelpunkt-Koordinaten aller vier Ränder vorher aufschreibst.

5 b) Deklariere außerdem ein Referenz-Attribut vom Typ STAUBSAUGER und initialisiere es im Konstruktor.

5 c) Überschreibe die geerbte Methode *tasteReagieren(...)*.
Bei Betätigung der Pfeil-Tasten (links, rechts) soll sich der Roboter jeweils um 10 Grad im bzw. gegen den Uhrzeigersinn drehen.

5 d) Überschreibe die geerbte Methode *tick()*. Bei jedem Aufruf soll sich der Roboter um einen Schritt bewegen.

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

Schritt 6: Chips einsammeln

Bisher reagiert der Staubsauger nicht auf die Chips.

6) Begib dich in die Methode *tick()* und dort unmittelbar unter die Zeile, welche den Roboter bewegt.
Füge dort für jeden Chip eine bedingte Anweisung ein, welche prüft, ob der Staubsauger einen Chip schneidet. Falls dies der Fall ist, so 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.

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 – sagen wir mal 50 Stück – dann wirst du im Deklarations- und auch im Initialisierungs-Teil 50 Mal denselben Code schreiben, bis auf die Nummer des Chips.

Ähnlich sieht es an der Stelle im Code aus, 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.

7 a) Lösche im Deklarations-Teil die drei Chips und deklariere stattdessen ein Array vom Typ CHIP. Nenne es *chips*.

7 b) Initialisiere im Konstruktor – genau über den drei Chips – zunächst das Array selbst mit der Länge 50.
Betrachte nun genau den Code, mit dem du die drei Chips bisher initialisiert hast. Versuche nun diese Idee in einer Zähl-Schleife umzusetzen, die alle 50 Chips „in einem Rutsch“ initialisiert.
Vergiss nicht, die drei alten Chips wieder aus dem JAVA-Code zu entfernen!

7c) Begib dich nun in die Methode *tick()* und versuche auch die Kollisions-Kontrolle des Roboters mit den Chips durch eine Zähl-Schleife „in einem Rutsch“ auf alle Chips im Array anzuwenden.
Vergiss auch hier nicht, den Code für die drei alten Chips zu entfernen!

7 d) Teste deine Veränderungen, indem du ein Objekt der Klasse ZIMMER erzeugst. Funktioniert alles wie erwartet nun auch mit 50 Chips?

Schritt 8: Künstliche Intelligenz

8) Programmiere den Roboter nun so, dass er den Raum nicht verlässt. Er soll, wenn er einen Rand schneidet zuerst etwas zurück fahren und sich eine andere Richtung suchen.
Auf diese Weise wird die Tastatur-Steuerung unnötig – der Roboter soll den Raum selbständig leer saugen.

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

Stoppe den Ticker, wenn der letzte Chip aufgesaugt wurde.
Setze hierzu eventuell neue Integer-Attribute ein.
Bei Taste N sollen die Chips zufällig neu verteilt und der Ticker neu gestartet werden.

Probiere verschiedene Strategien aus, wie der Roboter sich verhalten könnte, wenn er einen Rand oder einen Chip schneitet. Versuche auf diese Weise durch kreatives Probieren eine Strategie zu entwickeln bei welcher der Staubsauger das Zimmer möglichst schnell gesäubert hat.

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