Ein Notfallprogramm für den Roboter… Wir haben Millionen in die Hardware des Roboters und unendlich viel Zeit in die Implementierung der Software und seiner Verbesserungen gesteckt. Daher soll der Roboter ein Notfallprogramm bekommen mit dem er jederzeit den Ausgang wiederfinden kann, wenn er sich in einem zerstörten Kraftwerk befindet und der normale Rückweg plötzlich versperrt ist…

Die Roboter lernen zu zählen ...

ZIEL: *Variablen als Speicher für jeweils einen Wert kennen und einsetzen können. Variablen verwenden, um Roboter das Zählen beizubringen.*

Attribute beschreiben Eigenschaften und Zustände

Wir sehen die Roboter in ihrer Welt agieren. Wir sehen auch, wie viel Energie sie noch haben. Mit getEnergie() können wir das auch abfragen. Dann erhalten wir von ihm eine Zahl als Antwort. Woher weiß dieser Roboter aber, wie viel Energie er noch hat? Wie merkt er sich diese Zahl?

Wenn mehrere Roboter auf der Welt herum laufen, dann gibt jeder auf Nachfrage die für ihn richtige Antwort. Jeder Roboter führt darüber Buch. Jeder Roboter hat sein Gedächtnis. Dazu verwendet er eine Variable: energie – man sagt auch, dass jedes Roboterobjekt eine Eigenschaft energie hat. In dieser Variable merkt er sich die Energie, genauer: den aktuellen Wert der Energie, die ihm verbleibt. Dieser Wert kann immer nur eine ganze Zahl sein. Man sagt: Der Typ des Attributs ist integer (engl. für Ganzzahl). In Java wird dafür die Kennzeichnung int verwendet.

Den Wert kannst du z.B. um 15 verringern, indem du dem Roboter den Befehl gibst:

- verbraucheEnergie(15) — Damit hat er 15 Energiepunkte weniger, egal wie viele es vorher waren (jedoch unterschreitet er 0 Energiepunkte nicht).

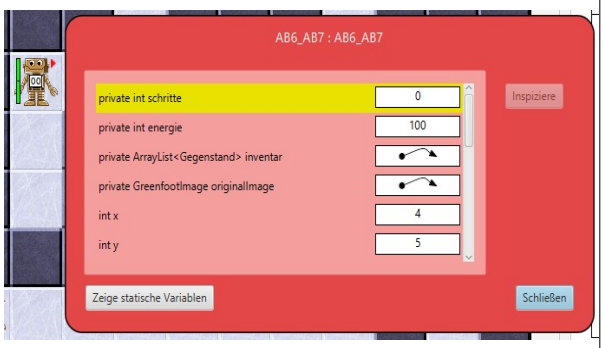
Den aktuellen Wert kannst du erfragen, indem du ihm die Anfrage stellst:

- getEnergie() — Er wird dir die Restenergie als Antwort nennen.

Die aktuellen Werte aller Attribute bestimmen den **Zustand** eines Roboters.

Aufgaben:

1. *Erprobe an zwei Robotern verbraucheEnergie(...) bzw. getEnergie() und kontrolliere mit INSPECT (Rechtsmausklick auf den Roboter → Inspizieren) den Zustand der Energie. Verschiebe ggf. das Inspect-Fenster so, dass es nicht mehr über der Roboterwelt liegt. Rechts siehst du einen Roboter und sein INSPECT-Fenster.*



2. *Notiere, welche weiteren Eigenschaften des Roboters in Attributen gespeichert sind. Nenne mindestens eine Methode, mit der Du die Werte der ganzzahligen Attribute ändern kannst.*

Bei den Integer-Attributen kann man die Werte direkt sehen. Bei komplexeren Attributen (z.B. inventar) wird nur ein Pfeil angezeigt. Die Attribute sind komplexer und nicht mit einer einzelnen Zahl auszudrücken. Programmieren musst du das noch nicht. Aber durch Klicken auf den Pfeil kannst du dir anschauen, was dahinter steckt. Du kannst beispielsweise erkennen, wo der Roboter speichert, wie viele Gegenstände er in seiner Inventarliste hat.

Aufgaben:

3. *Lasse dir durch einen Klick auf den Pfeil hinter inventar anzeigen, wie groß die Inventarliste ist. Versuche herauszufinden, wie groß das Bild (image) eines Roboters ist. Dazu musst du mehreren Pfeilen folgen.*

Jede Variable, die die Werte für eine Eigenschaft speichert, kannst du dir als beschriftete Kiste

vorstellen. Vorne steht der Name der Variable, damit die Kiste bzw. ihr Speicherplatz wiedergefunden werden kann. Namen von Variablen beginnen mit einem Kleinbuchstaben. Setzt sich der Name aus mehreren Wörtern zusammen, darf man keine Leerzeichen verwenden, sondern beginnt jedes neue Wort mit einem Großbuchstaben. Das bezeichnet man als Camel-Case.

In der Kiste gibt es zu jedem Zeitpunkt genau einen aktuell gültigen Wert. Mit set-Methoden (und auch durch andere Methoden) werden neue Werte in die Kiste gebracht; mit der get-Methode nach dem aktuellen Wert gefragt und dieser Wert dann dem Anfragenden als Ergebnis genannt. Dabei verbleibt der Wert unverändert in der Variable. So wie beim Anhören eines Musikstücks auf deinem Smartphone der Musikinhalt ja weiterhin gespeichert bleibt.

Der Typ der Variable legt die Größe der Kiste fest. In int-Kisten (32 Bit groß) passen z.B. ganze Zahlen zwischen -2.147.483.648 und +2.147.483.647. Für größere Zahlen bräuchte man long-Kisten (64 Bit groß). Dort passen Zahlen zwischen -9.223.372.036.854.775.808 und 9.223.372.036.854.775.807 hinein.

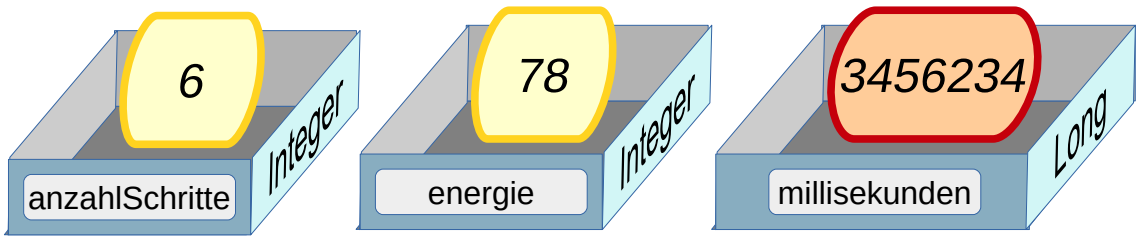


Abb. 1: int und long-Variablen als "Kisten" (Quelle: eigenes Werk)

Die Zeichnungen zeigen zwei Variable vom Datentyp int. In solche Variable können nur ganze Zahlen (= Integer; daher int) gesteckt werden. Auch wenn die Bilder nur positive Ganzzahlen suggerieren, sind Integer-Zahlen positive und negative Ganzzahlen.

Aufgaben:

4. ***Denk- und Sprechweisen**:*
   a) Wieso kann man sich Variablen als Kisten vorstellen?
   b) Welche der vier Sprechweisen hältst du für gut?
   1: Die Variable anzahl hat den Inhalt 7.
   2: Die Variable anzahl hat den Wert 7.
   3: Die Variable anzahl hat 7 Werte.
   4: anzahl ist 7.
5. ***Namensgebung**:*
   Schlage sinnvolle Namen für Attribute vor, die speichern
   a) wie viele Akkus der Roboter besitzt.
   b) wie viele Akkus der Roboter schon eingesetzt hat.
   c) wie viele Drehungen er gemacht hat. (Was genau gibt die gespeicherte Zahl dann an? Wie werden Links- bzw. Rechtsdrehungen gespeichert?)
   d) wie viele Brennstäbe er schon gefunden hat.

Eigene Attribute

Der Roboter AB6_AB7 hat eine neue Eigenschaft bekommen: Er hat das Attribut Schritte, das zählt wie viele Schritte er schon gemacht hat.

Dazu müssen drei Dinge erledigt werden:

- ✔ Das Attribut muss deklariert werden (die Kiste muss erschaffen werden): Oberhalb aller Methoden im Quelltext werden die Attribute angegeben. Die Deklaration beginnt mit private, dann folgt der Typ und der Name des Attributs.

  private int schritte;

- ✔ Der Wert des Attributs muss initialisiert werden (die Kiste muss einen sinnvollen Anfangswert erhalten): Dies macht man im sogenannten Konstruktor. Dies ist eine Methode mit dem gleichem Namen wie die Klasse (Achtung: Sie hat keinen Rückgabetyp, auch kein void). Sie wird automatisch aufgerufen, wenn das Roboterobjekt erzeugt wird. Dort wird der Wert von schritte auf 0 gesetzt. Um zu verdeutlichen, dass dieses Attribut zu dem Roboter gehört, schreibt man statt „schritte“ immer „this.schritte“:

  this.schritte = 0;

  Anmerkung: Wenn man Strg-Leerzeichen drückt, bekommt man alle verfügbaren Attribute (nur ab Version 3.1.0) und Methoden angezeigt.

- ✔ In der Methode einsVorMitZaehlen() wird der Wert des neuen Attributs um 1 erhöht: *this.schritte++;* und dann die normale einsVor()-Methode aufgerufen.

Aufgaben:

6. *Überprüfe im INSPECT-Fenster, ob der Roboter ordentlich seine Schritte zählt.*
7. ***Schrittezähler:** Implementiere eine Methode bisWandMitZaehlen() unter Verwendung von einsVorMitZaehlen(), die den Roboter bis zur Wand laufen lässt und dabei die Schritte zählt.*
8. ***Drehzähler:** Führe ein neues Attribut drehungen ein (deklariere und initialisiere das Attribut). Dieses soll bei jeder Rechtsdrehung um eins erhöht und bei jeder Linksdrehung um 1 erniedrigt werden (es gibt analog zu ++ auch - -). Führe dazu die beiden Methoden dreheLinksMitZaehlen() und dreheRechtsMitZaehlen() ein.*

Get-Methoden

Wie bei den anderen Eigenschaften auch, soll der Roboter auch ohne Inspect-Fenster Auskunft darüber geben können, welchen Wert seine Attribute haben. Dazu brauchen wir get-Methoden. Im letzten Kapitel hast du Methoden kennengelernt, die mit true oder false antworten. Nun möchten wir mit einer Zahl als Ergebnis antworten. Daher muss der Rückgabetyp der Methode int sein:

```
/** nennt die gesamten zurück-
 * gelegten Schritte */
public int getSchritte() {
    return this.schritte;
}
```

public int getSchritte() {…}

Statt return true oder return false, wollen wir nun den Wert des Attributs zurückgeben. Dazu verwenden wir einfach den Namen des Attributs:

return this.schritte;

Aufgaben:

9. ***Eine get-Methode:** Verändere die Methode getSchritte() im Roboter AB6_AB7, so dass die Methode nicht immer 0 zurück gibt, sondern den Wert des Attributs schritte. Warum kann die Methode dafür nicht lauten: public void ... ?*
   Warum enthält die Methode bisWandMitZaehlen() aus Aufgabe 7 keine Zeile, die mit return ... beginnt?
10. ***Deine get-Methode:** Implementiere eine get-Methode für die Anzahl der Drehungen. Teste*

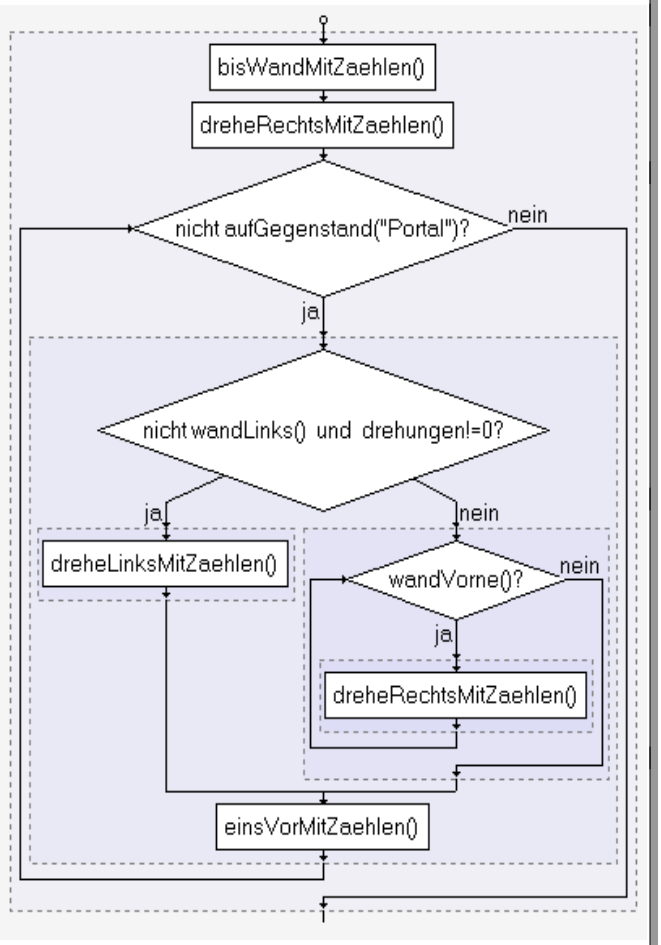
deine get-Methode.

Einsatz 6: Das Notfallprogramm wird getestet

Der Roboter wird nicht immer so gut informiert, wie sein Einsatzgebiet aussieht. Unsere hohen Entwicklungskosten sollen nicht verloren gehen, nur weil der Roboter irgendwann mal den Ausgang nicht wiederfindet. Daher bekommt der Roboter ein Not-Programm, mit dem er in beliebigen Umgebungen den Ausgang (gekennzeichnet durch das Portal) immer wiederfindet, egal wie viele Ecken und Wände im Weg sind.

Geht das wirklich? Ja, John Pledge, ein 12-jähriger Junge, hat entdeckt, wie das geht. Wir müssen dazu nur Drehungen zählen können.

Seine Idee war folgendermaßen: Im Prinzip folgt er permanent einer Wand so, dass seine linke Hand die Wand berührt. Da das aber zu Problemen führt, wenn die linke Hand beispielsweise an einer Säule ist, muss man irgendwann loslassen und geradeaus laufen. Das macht Pledge, wenn er sich genauso oft links wie rechtsrum gedreht hat.



Aufgabe:
Implementiere den Pledge-Algorithmus als Not-Programm (Methode einsatz6()). Der Roboter wird dann in einer unbekannten Umgebung ausgesetzt und muss das Portal finden.

Hinweis:
Im Bild rechts siehst du ein Flussdiagramm des Pledge-Algorithmus. Es besteht aus ein paar Anweisungen, die du gerade implementiert hast und nur aufrufen musst und einigen while-Schleifen und if-Anweisungen. Überlege dir zunächst, welche der Fragen/Bedingungen zu einer while-Schleife und welche zu einer if-Anweisung gehören. Woran erkennt man den Unterschied?

Hinweis: Korrigiere die Befehle (wie beispielsweise wandLinks()), falls nötig!

Hinweis: Wenn dein Programm nicht funktioniert, hast du vermutlich das Flussdiagramm nicht korrekt umgesetzt. Kopiere den Abschnitt deines Quelltextes über die Zwischenablage in das Programm "Flussdiagramme.exe" und vergleiche deine Version mit dem Original.

Bildquellen: Die verwendeten Bilder des Roboterszenarios sind alle ohne Bildnachweis verwendbar (selbst gezeichnet, Pixabay Lizenz oder Public Domain). Genaue Nachweise: siehe *bildquellen.html*.