



Hühner sind schlauer als viele vermuten. Besonders wenn DU die Hühner programmierst. Dann können sie sich sogar dauerhaft etwas merken und Fragen dazu beantworten...

Die Figuren merken sich was dauerhaft ...

ZIEL: Attribute als Speicher für jeweils einen Wert kennen und einsetzen können. Den Konstruktor einer Klasse kennenlernen und zum Initialisieren von Attributen verwenden können.

Attribute beschreiben Eigenschaften und Zustände

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

Natürlich mit einer Variablen. Die hast du ja schon in Scratch kennengelernt. Jedes Objekt verwaltet dabei seine Variablen selbst. Wenn also mehrere Figuren auf der Welt herum laufen, dann kann jede einen unterschiedlichen Wert für hunger speichern. Diese Variablen, in denen die Objekte etwas dauerhaft speichern, nennt man Eigenschaften oder Attribute des Objekts. Daneben gibt es auch lokale Variablen, die nur innerhalb einer Methode verwendet werden und danach wieder vergessen werden, und Parameter, die an eine Methode übergebene Werte speichern.

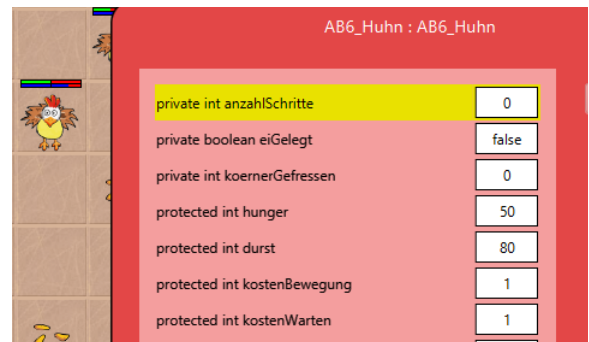
Den aktuellen Wert von Eigenschaften kann man in der Regel mit einer Methode abfragen, deren Name mit "get..." oder "gib..." beginnt, z.B.:

- `getHunger()` bzw. `getDurst()` wird dir die Restenergie als Antwort nennen.

Die aktuellen Werte aller Attribute bestimmen den **Zustand** einer Figur.

Aufgaben:

1. Rufe bei zwei Hühner das Inspect-Fenster auf (Rechtsmausklick auf das Huhn → Inspizieren) und vergleiche den Wert der Attribute hunger und durst mit der Lebensanzeige des Huhns. Lasse das Huhn laufen und beobachte die Werte der Attribute. Rechts siehst du ein Huhn und sein INSPECT-Fenster.
2. Notiere, welche weiteren Eigenschaften des Huhns in Attributen gespeichert sind. Nenne mindestens eine Methode, mit der Du den Wert eines Attributs ändern kannst.



Du hast gelernt, dass man sich Variablen als beschriftete Kiste vorstellen kann. Vorne auf der Kiste steht der Name der Variable, in der Kiste liegt ihr momentaner Wert. In Java gibt es die Vereinbarung, Namen von Variablen mit einem Kleinbuchstaben beginnen zu lassen. 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.

Außerdem muss man in Java bei der Deklaration der Variablen (=Erschaffen der Kiste) festlegen, was man in der Kiste speichern möchte (=Typ der Variablen). Je nach Typ wird

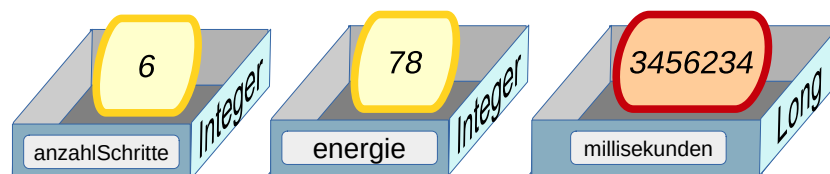


Abb. 1: int und long-Variablen als "Kisten" (Quelle: eigenes Werk)



unterschiedlich viel Speicherplatz reserviert. 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.

Art	ganze Zahl	ganze Zahl	Wahrheitswert	Kommazahl	Buchstabe	Text
Typ	int	long	boolean	double	char	String
Bsp	2837	9288376172	true/false	23.4	'a'	"Beispiel"

Aufgaben:

3. Denk- und Sprechweisen:

- Wieso kann man sich Variablen als Kisten vorstellen?
- Welche der vier Sprechweisen hältst du für gut?
 - Die Variable *anzahl* hat den Inhalt 7.
 - Die Variable *anzahl* hat den Wert 7.
 - Die Variable *anzahl* hat 7 Werte.
 - anzahl* ist 7.

4. Namensgebung:

Schlage sinnvolle Namen für Attribute vor, die speichern,

- wie viele Schritte ein Schaf gegangen ist.
- wie viele Drehungen es gemacht hat. (Was genau gibt die gespeicherte Zahl dann an? Wie werden Links- bzw. Rechtsdrehungen gespeichert?)
- wie viele Blumen es gefressen hat.

Eigene Attribute

Das **AB6_Huhn** hat eine neue Eigenschaft: Es hat das Attribut `koernerGefressen`, das zählt wie viele Körner ein Schaf im Laufe seines Lebens schon gefressen hat.

Dazu müssen drei Dinge erledigt werden. Das ist genauso wie bei Scratch:

1 Deklaration:

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. In Scratch hat man mit "Neue Variable" diesen Schritt ausgeführt.

```
private int anzahlKoerner;
```



2 Initialisierung:

Der Wert des Attributs muss initialisiert werden (die Kiste muss einen sinnvollen Anfangswert erhalten): Dies macht man im sogenannten **Konstruktor**. Er wird automatisch aufgerufen, wenn das Objekt erzeugt wird. Er ist die Methode mit dem gleichem Namen wie die Klasse (Achtung: Sie hat keinen Rückgabety, auch kein `void`). Dort wird der Wert von `anzahlKoerner` auf 0 gesetzt. In Scratch hat man das in der Regel direkt nach "Wenn grüne Flagge gedrückt" gemacht.

```
// setze anzahlKoerner auf 0
anzahlKoerner = 0;
```



3 Verändern des Attributs:

In der Methode `fresseMitZaehlen()` wird nach dem Fressen der Wert des neuen Attributs um 1 erhöht:

```
// ändere anzahlKoerner um 1
anzahlKoerner = anzahlKoerner + 1;
// neuer Wert = alter Wert + 1;
```



Aufgaben:

5. Überprüfe im INSPECT-Fenster, ob das Huhn ordentlich seinen Fresserfolg zählt.
6. **Schrittezähler:** Implementiere eine Methode `sucheKoerner()`, die das Huhn bis zum nächsten Feld mit Körnern (`istAuf("Koerner")`) laufen lässt und dabei die Schritte zählt. Anschließend soll das Huhn die gefunden Körner fressen (natürlich mit Zählen). Dekлариere dazu ein Attribut `entfernungZumNest`, initialisiere es im Konstruktor und zähle dann die Schritte. Überprüfe im INSPECT-Fenster, ob die Schritte richtig gezählt werden.
7. **Schrittanzeige:** Die Methode `gibAnzahlGefressenerKoerner()` gibt den Wert des Attributs zurück. Implementiere genauso eine Methode `gibEntfernungZumNest()`. Hinweis: Denke daran, dass diese Methode einen Wert zurückgeben soll. Was muss anstelle von `void` hier stehen?
8. **Schrittezähler 2:** Verbessere deine Methode `sucheKoerner()`, so dass sie auch funktioniert, wenn vor dem Huhn gar keine Körner liegen. Das Huhn soll dann so lange laufen, wie vor ihm frei ist.
9. **Heimwärts:** Implementiere die Methode `laufeZurueckZumNest()`, die das Huhn zurück zum eigenen Nest laufen lässt. Damit das Huhn nicht auf dem fremden Nest stehen bleibt, musst du dafür das Attribut `entfernungZumNest` nutzen. Teste auch die Methode `picken()` an allen Hühnern. Sie sollte nun mit Hilfe deiner Methoden ein Huhn alle Körner vor ihm aufpicken und danach zum Nest zurückkehren lassen. Hinweis: Wie sollte sich der Wert der Variable ändern, wenn das Huhn einen Schritt zurück Richtung eigenes Nest macht?
10. **Jedes Huhn legt ein Ei:** Steht ein Huhn auf einem Feld mit einem Nest, kann es versuchen ein Ei zu legen (`versucheEiZulegen()`). Der Erfolg ist umso größer, je besser genährt das Huhn ist. Implementiere die Methode `gackere()`, die das Huhn einen Versuch unternehmen lässt. War er erfolgreich (`istAuf("Ei")`), dann soll sich das Huhn dies dauerhaft merken. Führe dazu ein Attribut `eiGelegt` ein, das `true` ist, wenn das Huhn in seinem Leben schon einmal ein Ei gelegt hat. Andernfalls ist es `false`. Implementiere auch die dazu passende Methode `hatEiGelegt()`, die den Wert des Attributs zurück gibt. Teste deine Methoden.

Kommentare

Kommentare sind sinnvoll, um ein Programm lesbar zu machen. Man unterscheidet dabei einzeilige Kommentare (`// ...`) und Kommentare, die über mehrere Zeilen gehen (`/* ... */`):

```
// Kommentar einzeilig

int zahl; // auch hinter Quelltext möglich

/* Kommentar mehrzeilig
 * wenn man viel zu sagen hat.
 */
```

Aufgaben:

11. Füge in die Methode `leveltest6()` der Klasse `AB6_Bauer()` Kommentare ein, die erläutern, in welche Richtung der Bauer gerade läuft. Hinweis: Überlege dir, ob es sinnvoll ist, den Kommentar vor einem einzelnen `einsVor()` zu platzieren, oder ob eine ganze `while`-Schleife für die Bewegung in eine Richtung verantwortlich ist.



Leveltest 6: Hühnerbauernprüfung

Der AB6_Bauer soll zeigen, dass er erfolgreich einen Hühnerhof führen kann. Er muss dazu mindestens 30 Eier von seinen freilaufenden Bio-Hühnern einsammeln.

Um diese Aufgabe zu lösen, solltest du zunächst zwei Aufgaben lösen, die du noch in der Standardumgebung testen kannst.

Aufgabe 1:

Lasse den Bauer beim Laufen nach links zählen, wie viele Körner in dieser Reihe schon liegen. Beim Rückweg werden weitere Körner auf freien Feldern abgelegt

(`ablegen("Koerner")`), bis die Mindestzahl von 4 Körner pro Reihe erreicht ist.

Aufgabe 2:

Lasse den Bauer auf dem Rückweg zum Haus die in den Nestern liegenden Eier einsammeln (`aufnehmen();`) und zählen. Die Methode `getAnzahlEier()` muss die Zahl der gesammelten Eier zurückgeben.

Wenn du diese beiden Aufgaben gelöst hast, müsstest du die Hühnerbauernprüfung (`leveltest06`) schon bestehen. Du kannst versuchen, die dafür benötigte Zeit noch zu optimieren.

