JavaKarol

Die Fortführung von RobotKarol
zur Programmierung in Java
Version 3.0

Ulli Freiberger
freiberger@asv.bayern.de

Inhaltsverzeichnis

Einleitung

Robot Karol

Den Programmen „Robot Karol“ und „JavaKarol“ liegt die Idee von „Karel, the Robot“ zugrunde, wie sie zum ersten Mal in „Pattis, Richard E; Karel the Robot: A Gentle Introduction to the Art of Programming; John Wiley & Sons, 1981“ veröffentlicht wurde.

Das Grundkonzept ist, einen Roboter zu programmieren, der in einer „Bildschirmwelt“ lebt. Wenn die Programme laufen, sehen die Schülerinnen und Schüler an der Reaktion des Roboters sofort, was sie programmiert haben und ob ihr Programm die Aufgabenstellung erfüllt. Diese unmittelbare Rückmeldung ist gerade für Einsteiger extrem wertvoll.

Ein Roboter lebt bei „Robot Karol“ in einer dreidimensionalen Welt mit einem Boden aus quadratischen Grundpflastern und einer wählbare Höhe. In dieser Welt kann sich Karel von Quadrat zu Quadrat bewegen, Ziegelstapel aufbauen und abbauen, auf die Ziegelstapel klettern und Markierungen setzen.

Das Programm „Robot Karol“ ist eine Programmierumgebung mit:

- einer eigenen Programmiersprache RobotKarol, die Schülerinnen und Schüler den Einstieg in die Programmierung erleichtert, zur Einführung in die Algorithmik gedacht ist und erste Schritte einer objektorientierten Programmierung ermöglicht;
- einem Editor mit Syntaxhervorhebung und Schlüsselwort-Ergänzung;
- einem Interpreter der schrittweises Abarbeiten von RobotKarol-Programmen ermöglicht und diese Programme in einer Code-Übersicht zeigt;
- einer grafischen Darstellung der Welt, die den Roboter Karol als Figur im Raum zeigt und ihn je nach Anweisungen bewegt, ...
- einem Tool zum interaktiven Erstellen von Karolwelten.

Bezugsquelle für diese Programmierumgebung: http:/www.schule.bayern.de/karol

Karol-Roboter und ihre Welt

Ein Karol-Roboter kann durch ein Programm in einer Welt mit Quadratmuster bewegt werden. Es gibt ein Objekt der Klasse **WELT** und dieses hat die grundlegenden Eigenschaften Breite, Länge und Höhe. Eingerahmt ist die Welt an allen vier Seiten von Wänden in der entsprechenden Höhe.
Die Welt kann verschiedene Objekte aus den folgenden Klassen enthalten.

ZIEGEL: Diese Objekte kann ein Karol-Roboter vor sich hinlegen und später wieder aufheben. An einer Stelle sind mehrere Ziegel aufeinander möglich, jedoch maximal bis zur Höhe der Welt. Ein Karol-Roboter ist ein kräftiger Bursche und kann beliebig viele (oder begrenzte Anzahl, je nach Einstellung der RucksackGröße) Ziegel mit sich herumschleppen. Er kann auch auf Ziegelstapel hinaufspringen und herabspringen, jedoch maximal so hoch wie seine Eigenschaft „maximale Sprunghöhe“ dies zulässt. Ab der Version „Java Karol 3.0“ können die Ziegel die Farbe "rot" (Standard), "gelb",

"blau" oder "grün" annehmen.

MARKE: Diese Objekte kann ein Karol-Roboter an der Stelle anbringen, an der er sich gerade befindet. An jeder Stelle ist höchstens eine Marke möglich. Eine Stelle kann markiert sein oder nicht.
Ab der Version „Java Karol 3.0“ können die Marken die Farbe "rot", "gelb" (Standard), "blau", "grün" oder "schwarz" annehmen.

QUADER: Mit Quader kann die Welt zusätzlich gestaltet werden. An einer Stelle kann höchstens ein Quader stehen und ein Karol-Roboter kann nicht auf einen Quader springen. Im Programm verhalten sich Quader wie eine Wand.

Ein Karol-Roboter ist aus objektorientierter Sicht ein Objekt der Klasse **ROBOTER**, das in einem Objekt der Klasse WELT enthalten ist. Ein Roboter hat folgender Datenstruktur:

Attribute (Eigenschaften): Position (zweidimensional mit X und Y); Blickrichtung; maximale Sprunghöhe; Anzahl Ziegel im Rucksack; RucksackGröße;

Methoden (Fähigkeiten): Vom „Herstellungswerk“ wird der Roboter mit einer Reihe von vordefinierten Methoden ausgeliefert, mit denen er bestimmte Vorgänge ausführen kann. Ein Roboter hat auch Methoden, mit denen er auf eine Anfrage mit WAHR oder FALSCH antwortet (zum Beispiel „stehst du vor der Wand?“).

Durch Methodenaufrufe kann sich der **Roboter in der Welt bewegen**. Er kann:

- vorwärts gehen und sich auf der Stelle drehen;
- auf Ziegel hinaufsteigen, so weit seine Sprunghöhe dies zu lässt (Defaulteinstellung einen Ziegel hoch);
- von Ziegel herabsteigen (maximal die Sprunghöhe);
- Ziegel vor sich hinlegen oder aufheben; ist die RucksackÜberprüfung ausgeschalten (Default), so kann er beliebig viele Ziegel hinlegen bzw. aufheben;
- Marken unter sich setzen oder löschen;
- Quader setzen und aufheben, aber nicht besteigen;
- auf Fragen bezüglich seiner Position antworten.

Zusätzlich zu den „vom Werk vorgesehenen“ Methoden können in einer Unterklasse zu der Klasse ROBOTER **neue Eigenschaften und Methoden** hinzugefügt werden. Karol-Roboter die von dieser Unterklasse abstammen verfügen dann über diese zusätzlichen Fähigkeiten.

Aus **zustandsorientierter** Sicht kann das JavaKarol-System eine feste Anzahl von verschiedenen **Zuständen** annehmen. Ein Zustand des JavaKarol-Systems wird beschrieben durch:

- die Breite, Länge und Höhe der Welt
- die Anzahl und Position der Ziegel, Quader und Marken
- die Position und Blickrichtung von Karol-Roboter
- die weiteren Eigenschaften von Karol-Roboter. Diese können, bei entsprechender Einstellung, auch die Anzahl der Ziegel im Rucksack und die Rucksackgröße umfassen.

Vor dem Programmstart befindet sich die Karol-Welt in einem frei festlegbaren **Ausgangszustand**. Ein Karol-Programm beschreibt einen **Zustandsübergang** von diesem Ausgangszustand zu einem **Endzustand**, dabei werden meist viele Zwischenzustände eingenommen. Ausgelöst wird der Zustandsübergang durch den Programmstart. Unterschiedliche Ausgangszustände können zu unterschiedlichen Endzuständen führen. Man sieht das sofort, wenn dasselbe Programm auf unterschiedliche Ausgangszustände angewandt wird.

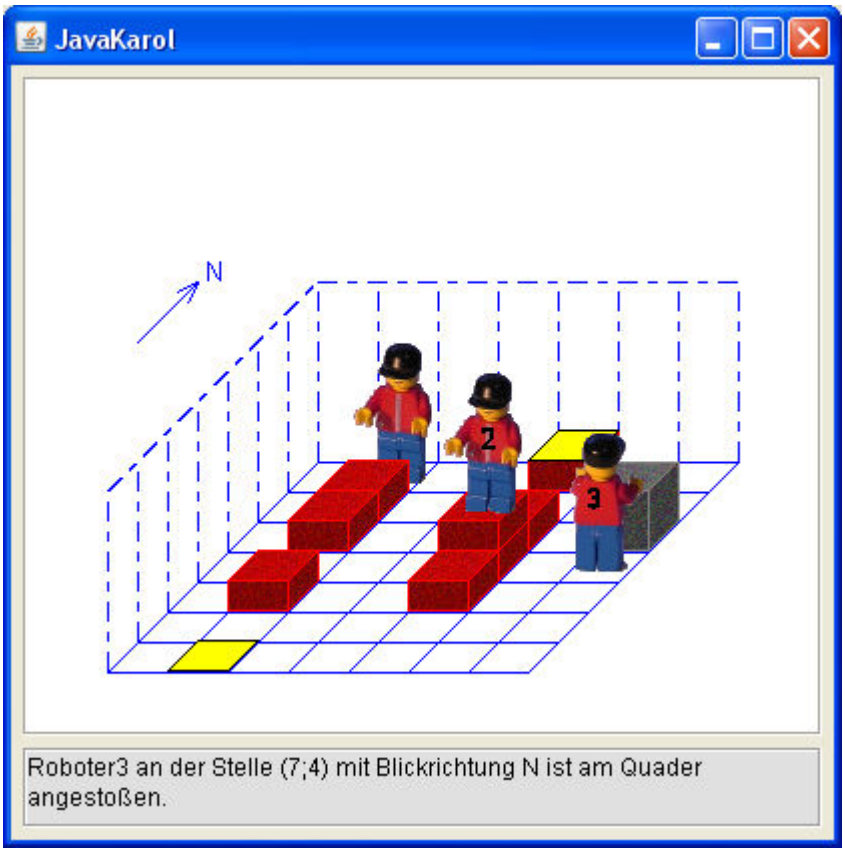
Der Zustand einer Welt, d. h. die Welt und die darin enthaltenen Objekte, kann in einer **Karolwelt-Datei** gespeichert werden. Die übliche Dateiendung für Karol-Welten ist *.kdw (Karol deutsch Welt). Für JavaKarol gibt es eine Erweiterung dieses Dateiformats mit der

Endung *.jkw (JavaKarol Welt). Beide Dateiformate sind untereinander kompatibel. In einer JavaKarolWelt-Datei werden auch die Zustände von mehr als einem Roboter abgespeichert. Karolwelt-Dateien mit einem KarolRoboter können einfach mit der Programmierumgebung RobotKarol erstellt werden.

JavaKarol

JavaKarol ist eine konsequente Fortführung von RobotKarol für die Programmiersprache **Java**. Hierbei finden dieselben Objekte wie bei RobotKarol Verwendung, so dass die Anwender sofort mit dem System vertraut sind. Die Programme werden jedoch jetzt in Java formuliert und nicht mehr in der speziellen Sprache RobotKarol. War diese Sprache zum Einstieg in die Programmierung sehr hilfreich und unerlässlich, so stehen den Schülerinnen und Schülern im fortgeschrittenen Informatikunterricht jetzt die gesamten Möglichkeiten der mächtigen, objektorientierten Programmiersprache Java zur Verfügung. Von einem direkten Einstieg mit JavaKarol im Anfängerunterricht wird aus didaktischen Überlegungen abgeraten (Java ist für eine erste Begegnung mit einer Programmiersprache viel zu komplex).

Da die Methoden der Roboter und das Verhalten der Roboter in der Welt aus RobotKarol übernommen wurden, ist es für die Schülerinnen und Schüler ein Leichtes, die im Anfangsunterricht erstellten Karol-Programme in die neue Notation zu übertragen. Darüber hinaus bietet JavaKarol weitere Features (wie zum Beispiel den Einsatz mehrerer Roboter in einer Welt), die eine Vertiefung in objektorientierter Modellierung/Programmierung unterstützen.



In einer Java-Bibliothek (javakarol.jar) werden die Klassen **Roboter** und **Welt** zur Verfügung gestellt. Die Klasse Welt verwaltet die Objekte, die zur Welt gehören sowie die Darstellung der Welt in einem Grafikfenster (**JavaKarol Fenster**).

Beim Neuanlegen eines Objekts der Klasse Welt wird automatisch ein Fenster geöffnet. In der Zeichenfläche werden die Objekte dargestellt, die in der Welt enthalten sind (d. h. Marken, Ziegel, Quader und Roboter). Bei einer Veränderung der Welt durch Roboteraktionen wird der Inhalt der Zeichenfläche aktualisiert. In manchen Situationen kann der Aufruf einer Robotermethode zu einem Programmabbruch führen (z. B. „an die Wand gelaufen“). In diesen Fällen wird im unteren Textfeld des Fensters eine Erklärung ausgegeben. Durch Schließen des Fensters über die Schaltfläche rechts oben wird das Programm beendet.

Karolwelten als Ausgangssituation für ein Programm können durch die Ausführung mehrerer Robotermethoden generiert werden. Das ist aber sehr aufwändig. Es wird empfohlen diese Welten mit der Programmierumgebung RobotKarol zu erstellen und als Karolwelt-Dateien (*.kdw) zu speichern. Diese Weltdefinitionen können dann als Grundlage bei der Erzeugung einer neuen Welt (d.h. eines Objekts der Klasse Welt) verwendet werden.

Durch Methoden der Klasse Welt können JavaKarol-Welten gespeichert werden (*.jkw).

Klassen in JavaKarol

Die Java-Bibliothek javakarol.jar enthält zwei öffentlich zugängliche Klassen: **Welt** und **Roboter**. Beide Klassen können in einem Java-Programm verwendet werden (`import javakarol.Welt` bzw. `import javakarol.Roboter`).

Positionsangaben in der Karolwelt erfolgen in einem Koordinatensystem mit X: 1 … Breite der Welt und Y: 1 … Länge der Welt. Auf einem Quadratfeld der Welt kann immer nur ein Roboter stehen.

Klasse Welt

Konstruktoren

Mindestens ein Weltobjekt muss instanziiert werden bevor ein Roboter angelegt werden kann. Welt Objekte können erzeugt werden durch direkte Größenangaben oder durch Angabe einer Karolwelt-Datei, die die Grundlage für das Welt-Objekt bildet.
Voraussetzung für eine erfolgreiche Einrichtung eines Weltobjekts ist das Vorhandensein der Grafiksymbole für Ziegel, Marke, Quader und die verschiedenen Roboter in der Bibliothek javakarol.jar (siehe auch Kapitel Roboterfiguren).
Ein Fehler bei der Instanziierung führt zu einem Programmabbruch (RuntimeException).

Konstruktor	Beschreibung	mögliche Ablauffehler
`Welt(int breite, int laenge, int hoehe)`	Erzeugen einer neuen, leeren Welt in der angegebenen Größe. Die Welt legt automatisch das zugehörige Anzeigefenster an. @param: **breite** Breite der Welt 1.. ; entspricht X-Koordinate **laenge** Länge der Welt 1.. ; entspricht Y-Koordinate **hoehe** Höhe der Welt 1..31	eines der Grafiksymbole kann nicht geladen werden;
`Welt(String weltdatei)`	Erzeugen einer neue Welt aufgrund einer Karolwelt-Datei mit einem vorgegeben Dateinamen. Die Einstellungen der Welt und der darin enthaltenen Ziegel, Quader und	das Datenformat der angegebenen Datei ist falsch; das Dateiauswahlfenster wurde mit „Abbrechen“

Konstruktor	Beschreibung	mögliche Ablauffehler
	Markierungen werden aus der Karolwelt-Datei geladen und die entsprechenden Objekte erzeugt. Es sind sowohl Karolwelten von RobotKarol (*.kdw) als auch von JavaKarol (*.jkw) möglich. Die Welt legt automatisch das zugehörige Anzeigefenster an. @param: **weltdatei** Dateiname der Karolwelt-Datei mit relativem oder absolutem Pfad und Dateiendung. Falls die Datei nicht vorhanden ist oder sich nicht im Klassenpfad der Anwendung befindet wird ein Dateiauswahlfenster geöffnet.	beendet; eines der Symbolbilder kann nicht geladen werden;
`Welt()`	Erzeugen einer neuen Welt aus einer Karolwelt-Datei ohne Angabe eines Dateinamens. Es wird immer erst ein Dateiauswahlfenster geöffnet, in dem die Karolwelt-Datei ausgewählt werden kann. Die Einstellungen der Welt und der darin enthaltenen Ziegel, Quader und Markierungen werden aus dieser Karolwelt-Datei geladen und die entsprechenden Objekte erzeugt. Es sind sowohl Karolwelten von RobotKarol als auch von JavaKarol möglich. Die Welt legt automatisch das zugehörige Anzeigefenster an.	das Dateiauswahlfenster wurde mit „Abbrechen“ beendet; eines der Symbolbilder kann nicht geladen werden;

Methoden

Methode	Beschreibung	mögliche Ablauffehler
`Speichern(String dateiname)`	Die Einstellungen der Welt zusammen mit den Positionen der Ziegel, Quader, Markierungen und Roboter wird gespeichert. Das Dateiformat ist stets jkw (JavaKarolWelt). Dieses Dateiformat ist eine Erweiterung des Dateiformats kwd (KarolWeltDeutsch) von RobotKarol.	das Dateiauswahlfenster wurde mit „Abbrechen“ beendet;

Methode	Beschreibung	mögliche Ablauffehler
	@param: **dateiname** Name der Karolwelt-Datei die erstellt werden soll; auch "" möglich.	
BildSpeichern (String dateiname)	Die Welt und die darin enthaltenen Grafikobjekte werden als Bild gespeichert. Mögliche Dateiformate sind bmp und jpg. @param: **weltdatei** Dateiname der Bilddatei die erstellt werden soll.	das Dateiauswahlfenster wurde mit „Abbrechen“ beendet; die Dateiendung ist weder „bmp“, „png“ oder „jpg“
ZurueckSetzen()	Die Welt und die darin enthaltenen Objekte werden wieder in den Startzustand (wie bei der Instanzierung) zurücksetzen.	die Karolwelt-Datei wurde in der Zwischenzeit in der Größe verändert;
ZiegelVerstreuen (int anzahlZiegel, int maxStapelhoehe)	Eine vorgegebene Anzahl an Ziegel wird zufällig über die Welt verstreut, wobei eine maximale Stapelhöhe eingehalten wird. @param: **anzahlZiegel** Anzahl an Ziegel die verstreut werden. **maxStapelhoehe** maximale Höhe die dabei ein Stapel annehmen kann.	

Klasse Roboter

Konstruktoren

Vor dem Erzeugen eines Roboter-Objekts muss ein Welt-Objekt angelegt sein, in dem der Roboter „lebt“.

Konstruktor	Beschreibung	mögliche Ablauffehler
Roboter(int startX, int startY, char startBlickrichtung, Welt inWelt)	Erzeugen eines neuen Roboters mit vorgegebener Startposition. Vorher muss ein Objekt der Klasse Welt angelegt werden, in der der Roboter lebt. @param: **startX** X-Koordinate der Startposition mit 1...WeltBreite	Welt Objekt ist noch nicht angelegt; an dieser Stelle steht bereits ein Roboter; an dieser Stelle steht ein Quader; die maximale Anzahl an Roboter wurde überschritten

Konstruktor	Beschreibung	mögliche Ablauffehler
	startY Y-Koordinate der Startposition mit 1...WeltLänge **startBlickrichtung** Startblickrichtung 'S','W','N','O' **inWelt** Referenz auf die Welt in der der Roboter leben soll	(z.Zt. 9);
`Roboter(Welt inWelt)`	Erzeugen eines neuen Roboters aufgrund der Einstellungen in einer Karolwelt-Datei. Vorher muss ein Objekt der Klasse Welt angelegt werden, in der der Roboter lebt. Wenn dieses Weltobjekt aus einer Karolwelt-Datei geladen wurde, dann werden die dortigen Roboterein-stellungen verwendet, sonst ein Default-Roboter mit Position (1,1) und Blickrichtung 'S' @param: **inWelt** Referenz auf die Welt in der der Roboter leben soll	Welt Objekt ist noch nicht angelegt; an dieser Stelle steht bereits ein Roboter; an dieser Stelle steht ein Quader; die maximale Anzahl an Roboter wurde überschritten (z.Zt. 9);

Attribute

Attribut	Beschreibung	Datentyp
`positionX`	X Position des Roboters	int
`positionY`	Y Position des Roboters	int
`blickrichtung`	Blickrichtung des Roboters	char; 'S', 'O', 'N', 'W'
`meineWelt`	Referenz auf die Welt in der sich der Roboter befindet	
`sprunghoehe`	maximale Sprunghöhe des Roboters	int
`rucksackInhalt`	Anzahl der Ziegel im Rucksack (*)	int
`rucksackMaximum`	maximales Fassungsvermögen des Ruck-sacks (*)	int
`kennung`	Eindeutige Kennung des Roboters in der Ka-rolwelt (1..9). Kennung 0 bedeutet nicht sichtbar.	int
`sichtbar`	Das Grafiksymbol eines unsichtbaren Robo-ters wird nicht in der Karolwelt gezeichnet. Er ist nicht mehr zu sehen und hat keine Wirkung auf die anderen Roboter.	boolean

(*) nur bei eingeschalteter Kontrolle der Tragfähigkeit ist der Wert gültig

Setzen-Methoden

Ein direktes Setzen der Positionsattribute ist nicht möglich. Über die Setzen-Methoden können einige Grundeinstellungen des Roboters festgelegt werden.

Methode	Beschreibung	mögliche Ablauffehler
`VerzoegerungSetzen(int msec)`	Setzen der Verzögerung. Nach jedem Methodenaufruf des Roboters wird um die angegebene Zeit (in Millisekunden) gewartet. Defaultwert 300 @param: **msec** Verzögerungszeit in Millisekunden	
`SprunghoeheSetzen(int neueHoehe)`	Setzen der maximalen Sprunghöhe des Roboters. Defaultwert 1 @param: **neueHoehe** Sprunghöhe in Ziegeleinheiten	
`RucksackMaximumSetzen(int maxZiegel)`	Maximales Fassungsvermögen des Rucksacks setzen. Durch Setzen auf einen Wert größer 0 wird die Prüfung des Rucksackinhaltes eingeschalten. Bei jedem Hinlegen und Aufheben von Ziegeln wird dann der Rucksackinhalt geprüft. Durch Setzen auf 0 wird die Rucksackprüfung wieder ausgeschaltet. Defaultwert 0 @param: **maxZiegel** Maximale Anzahl der Ziegel im Rucksack.	
`UnsichtbarMachen()`	Der Roboter wird bei der Welt abgemeldet. Ab diesem Zeitpunkt wird sein Grafiksymbol nicht mehr in der Karolwelt angezeigt. Er ist nicht mehr zu sehen und hat keine Wirkung auf die anderen Roboter.	
`SichtbarMachen()`	Der Roboter wird bei der Welt wieder angemeldet. Ab diesem Zeitpunkt wird sein Grafiksymbol wieder in der Karolwelt angezeigt.	an dieser Stelle steht bereits ein Roboter; an dieser Stelle steht ein Quader;

Methoden für Aktionen

Die Ausführung einer Anweisung bedeutet eine Botschaft an den Roboter schicken, der mit der entsprechenden Methode reagiert und dabei eine gewisse Aktion ausführt. Nicht ausführbare Methoden verursachen einen Programmabbruch (RuntimeException).

Methode	Aktion, die der Roboter ausführt	mögliche Ablauffehler
`Schritt()`	macht einen Schritt in die Blickrichtung	steht vor der Wand; steht vor einem Quader; steht vor einem anderen Roboter; kann nicht so hoch springen;
`LinksDrehen()`	dreht sich nach links (um 90°)	
`RechtsDrehen()`	dreht sich nach rechts (um 90°)	
`Hinlegen()`	legt vor sich einen roten Ziegel hin	steht vor der Wand; steht vor einem Quader; steht vor einem anderen Roboter; maximale Stapelhöhe erreicht; hat nichts zum Hinlegen(*);
`Hinlegen(String farbe)`	legt vor sich einen Ziegel der angegebenen Farbe hin @param: **farbe** "rot", "gelb", "blau", "grün"	steht vor der Wand; steht vor einem Quader; steht vor einem anderen Roboter; maximale Stapelhöhe erreicht; kennt die Ziegelfarbe nicht; hat nichts zum Hinlegen(*);
`Aufheben()`	hebt einen Ziegel auf, der vor ihm liegt (egal welche Farbe)	steht vor der Wand; steht vor einem Quader; steht vor einem anderen Roboter; kein Ziegel vor dem Roboter; maximales Tragvermögen erreicht(*);
`MarkeSetzen()`	setzt an seiner Position eine gelbe Marke	

Methode	Aktion, die der Roboter ausführt	mögliche Ablauffehler
MarkeSetzen (String farbe)	setzt an seiner Position eine Marke der angegebenen Farbe @param: **farbe** "rot", "gelb", "blau", "grün", "schwarz"	kennt die Markerfarbe nicht;
MarkeLoeschen()	löscht an seiner Position eine Marke (egal welche Farbe)	
QuaderAufstellen ()	stellt vor sich einen Quader auf	steht vor der Wand; steht vor einem Quader; steht vor einem anderen Roboter; vor ihm liegen schon Ziegel; vor ihm ist eine Marke;
QuaderEntfernen ()	entfernt einen Quader, der vor ihm steht	steht vor der Wand; steht vor einem anderen Roboter; vor ihm ist kein Quader;
Warten(float dauer)	wartet „dauer"-viele Sekunden	
TonErzeugen()	gibt einen Ton von sich	
MeldungAusgeben (String was)	übergibt eine Meldung an das Weltfenster, das diese Meldung im Textfeld darstellt	

(*) nur bei eingeschalteter Kontrolle der Tragfähigkeit

Logische Abfragefunktionen

Die Ausführung einer Bedingung bedeutet eine Anfrage an den Roboter schicken, der mit der entsprechenden Methode reagiert, die Situation seiner Umgebung begutachtet und mit true oder false antwortet. Die Negationen in der Sprache RobotKarol werden in JavaKarol durch den Not-Operator von Java realisiert (z. B. `!karol.IstWand()`)

Abfragefunktion	der Roboter meldet true,
IstWand()	wenn er vor der Wand oder vor einem Quader steht und in diese Richtung schaut
IstZiegel()	wenn er vor einem Ziegel (oder Ziegelstapel) beliebiger Farbe steht und zu diesem schaut
IstZiegel(String farbe)	wenn er vor mindestens einem Ziegel der angegebenen Farbe steht und zu diesem schaut @param: **farbe** "rot", "gelb", "blau", "grün"
IstZiegelLinks()	wenn links von ihm ein Ziegel oder Ziegelstapel beliebiger Farbe steht

Abfragefunktion	der Roboter meldet true,
IstZiegelRechts()	wenn rechts von ihm ein Ziegel oder Ziegelstapel beliebiger Farbe steht
IstMarke()	wenn er auf einer Marke beliebiger Farbe steht
IstMarke(String farbe)	wenn er auf einer Marke der angegebenen Farbe steht @param: **farbe** "rot", "gelb", "blau", "grün", "schwarz"
IstRoboter()	wenn er vor einem anderen Roboter steht und zu diesem schaut
IstRoboterInSicht()	wenn in seiner Blickrichtung ein anderer Roboter steht, egal wie weit weg. Ziegel und Quader behindern die Sicht.
IstBlickSueden() **IstBlickNorden()** **IstBlickWesten()** **IstBlickOsten()**	wenn er in diese Richtung schaut

Diese Abfragefunktionen sind nur sinnvoll, wenn die Überwachung der Tragfähigkeit des Roboters eingeschaltet ist (siehe RucksackMaximumSetzen()):

Abfragefunktion	der Roboter meldet true,
IstRucksackVoll()	wenn er seine maximale Tragfähigkeit erreicht hat
IstRucksackLeer()	wenn er keinen Ziegel mit sich trägt
HatZiegelImRucksack()	wenn er mindestens einen Ziegel mit sich trägt

Abfragefunktionen

Diese Abfragefunktionen unterstützen die Programmentwicklung.

Methode	Beschreibung	Datentyp des Rückgabewertes
AnzahlZiegelVorneGeben()	Gibt die Anzahl der Ziegel, die vor dem Roboter stehen	int
RoboterVorneKennungGeben()	Gibt die Kennung des Roboters, der vor dem Roboter steht. Steht keiner davor wird 0 zurückgegeben.	int

Geben-Methoden

Die Geben-Methoden sollten nur sehr zurückhaltend eingesetzt werden.

Methode	Beschreibung Gibt den Wert von	Datentyp des Rückgabewertes
PositionXGeben()	positionX	int
PositionYGeben()	positionY	int

Methode	Beschreibung Gibt den Wert von	Datentyp des Rück-gabewertes
`BlickrichtungGeben()`	blickrichtung	char
`SprunghoeheGeben()`	sprunghoehe	int
`KennungGeben()`	kennung Jeder Roboter hat eine eindeutige Kennung in dem Weltsystem.	int
`SichtbarkeitGeben()`	sichtbar	boolean
`AnzahlZiegelRucksackGeben()`	rucksackInhalt	int

Roboterfiguren

Zur 3D-Darstellungen eines Roboters werden vier Bilder - je nach Blickrichtung - verwendet. Für jeden Roboter gibt es unterschiedliche Bilder, so dass bei maximal 9 Robotern 36 Bilder erforderlich sind. Hinzu kommt jeweils ein Bild für Ziegel, Quader und Marke (je nach Farbe).
Alle Bilder sind in der Java-Bibliothek javakarol.jar im Ordner imgs abgelegt.

Die Bilder müssen GIF-Dateien mit 256 Farben sein, wobei die weiße Farbe transparent gesetzt ist.

Man kann auch eigene Roboter-Bilder verwenden. Diese müssen ebenfalls GIF-Dateien sein mit den Maßen 40 Pixel breit und 65-95 Pixel hoch. Für jede Blickrichtung muss ein entsprechendes Bild erstellt werden. Diese Bilder müssen dann gegen die entsprechenden Bilder in der Bibliothek javakarol.jar ausgetauscht werden.

Umsetzung RobotKarol nach Java

Die Umsetzung eines Karol-Programms von der Sprache RobotKarol in die Sprache Java erfordert keinen großen Aufwand. Die Bezeichner der Roboter-Methoden sind nahezu gleichgeblieben und das Verhalten der Roboter ist identisch. Da in JavaKarol mehr als ein Roboter möglich ist muss vor jeder Methode in Punktnotation das Objekt angegeben werden, dessen Methode aufgerufen werden soll (in RobotKarol gab es nur einen Roboter karol). Alle Kontrollstrukturen müssen in der Notation von Java formuliert werden.

Kontrollstrukturen

Name	Notation in RobotKarol	Notation in Java

einseitige bedingte Anweisung	**wenn** *Bedingung* **dann**
	  Anweisung
	  …
	***wenn**
		if (*Bedingung***)**
		{
		  Anweisung;
		  …
		}

zweiseitige bedingte Anweisung	**wenn Bedingung dann**
	  Anweisung
	  …
	sonst
	  Anweisung
	  …
	***wenn**
		if (*Bedingung***)**
		{
		  Anweisung;
		  …
		}
		else
		{
		  Anweisung;
		  …
		}

Wiederholung mit fester Anzahl	**wiederhole** *Anzahl* **mal**
	  Anweisung
	  …
	***wiederhole**
		for (int i=0; i<*Anzahl***; i++)**
		{
		  Anweisung;
		  …
		}

Wiederholung mit Anfangsbedingung	**wiederhole solange** *Bedingung*
	  Anweisung
	  …
	***wiederhole**
		while (*Bedingung***)**
		{
		  Anweisung;
		  …
		}

Wiederholung mit Endbedingung	**wiederhole**
	  Anweisung
	  …
	***wiederhole solange** *Bedingung*
		do
		{
		  Anweisung;
		  …
		}
		while (*Bedingung***)**

Einsatz von JavaKarol

Dateien

Das Paket JavaKarol wird als ZIP-Datei `JavaKarol30.zip` ausgeliefert. Bezugsquelle und Informationen unter `http://www.schule.bayern.de/karol` (Weiterleitung zu `https://www.mebis.bayern.de/infoportal/faecher/mint/inf/java-karol/`)

Zur Benutzung muss das Paket, einschließlich der Ordnerstruktur, in ein Verzeichnis auf dem Rechner entpackt werden, das in diesem Handbuch mit <JavaKarol_Verzeichnis> bezeichnet wird (z. B. `C:\Programme\JavaKarol`).

Das Paket JavaKarol umfasst folgende Dateien:

javakarol.jar	Java-Bibliothek mit den Verzeichnissen javakarol und imgs
	Das Verzeichnis javakarol enthält die Java-Klassen: Roboter.class, Welt.class, WeltAnzeige2D.class, WeltAnzeige3D.class, WeltFenster.class, FehlerAnzeige.class
	Das Verzeichnis imgs die GIF-Dateien für die Grafiksymbole: Marke.gif, Marke_*Farbe*.gif, Quader.gif, Ziegel.gif, Ziegel_*Farbe*.gif sowie für die Roboter robotNx.gif, robotSx.gif, robotWx.gif, robotOx.gif
JavaKarol30Handbuch.pdf	dieses Handbuch
Doc\	Javadoc Dokumentation der Klassen Roboter und Welt.
Beispiele\	einige Beispiele für den Einsatz von JavaKarol (jeweils *.java und *.class).
	Die meisten Beispiele stammen von RobotKarol und haben den gleichen Namen wie dort (nur mit dem Vorspann Beispiel_). Eine Beschreibung findet man im Handbuch von RobotKarol.
	Die Beispiele erwarten im Verzeichnis Karolwelten eine gleichnamige Karolwelt-Definitionsdatei.
Beispiele\Karolwelten	einige Karol-Welten *.kdw bzw. *.jkw. Die meisten Karol-welten stammen von der Programmierumgebung RobotKarol.
BlueJ\BlueKarol	Standardprojekt für die Verwendung von BlueJ. Dient als Grundlage und Vorlage für alle Karol-Projekte unter BlueJ. Enthält die Unterklassen ROBOTER und WELT.
BlueJ\Beispiele	Beispielprojekte für die Verwendung von JavaKarol unter der Entwicklungsumgebung BlueJ.

Copyright

Einsatz mit Java

Die Java-Bibliothek javakarol.jar kann zur Erstellung von Karol-Programmen mit jeder beliebigen Entwicklungsumgebung für Java verwendet werden. Sei es Eclipse, NetBeans, Java-Editor usw. oder ein Texteditor unter anschließender Verwendung des Javacompilers javac. Einzig beim Compilieren und beim Ausführen muss die Bibliothek im Klassenpfad von Java liegen.

Einige Hinweise bei direkter Verwendung von javac und java im Befehlsfenster:

1) Damit die Java-Klassen Roboter und Welt im Programm verwendet werden können muss der Programmtext *xyz*`.java` die Angaben
```
import javakarol.Roboter
import javakarol.Welt
```
enthalten.

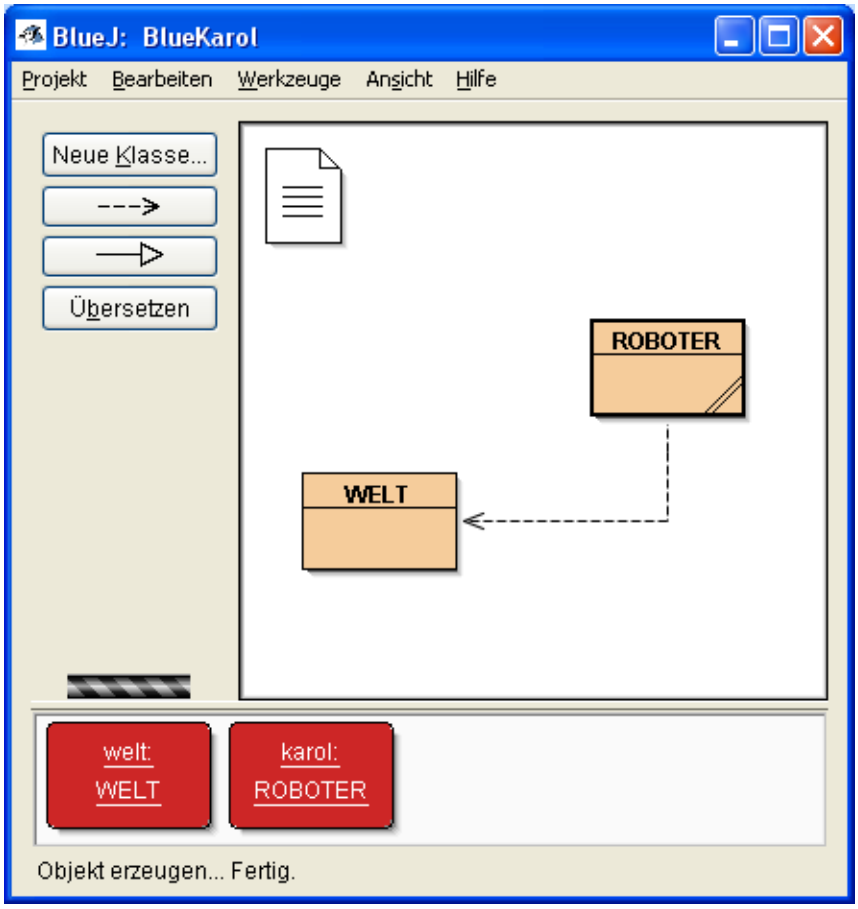
2) Zum Compilieren muss die Bibliothek `javakarol.jar` im Klassenpfad liegen (entweder über die globale Variable classpath oder über Optionsangabe beim Aufruf des Java-Compilers).
Z. B. Im selben Verzeichnis wie der Programmtext *xyz*`.java`:
```
javac  -classpath javakarol.jar;  xyz.java
```
Es wird eine Java-Klasse *xyz*`.class` erzeugt.

3) Zum Ausführen der compilierten Java-Klasse *xyz*`.class` muss beim Start der virtuellen Java-Engine ebenfalls die Bibliothek `javakarol.jar` im Klassenpfad liegen.
Z. B. Im selben Verzeichnis wie die Java-Klasse *xyz*`.class`:
```
java  -classpath javakarol.jar;  xyz
```

Einsatz mit BlueJ

BlueJ ist eine **Entwicklungsumgebung** für objektorientierte Java-Programme, die speziell für den Schulunterricht entworfen wurde. Es wurde vom BlueJ Team an der Deakin Universität in Melbourne, Australien, und der Universität von Kent in Canterbury, Großbritannien, konzipiert und umgesetzt.

Mehr Informationen über BlueJ sind im Web unter `https://www.bluej.org` verfügbar, sowie im Buch „Java lernen mit BlueJ“ (David J. Barnes, Michael Kölling; Pearson Studium; 6. Auflage 2017).



Das BlueJ-Fenster gliedert sich in drei Teilbereiche.

Links befindet sich ein grau unterlegter Bereich mit einigen Schaltflächen. Mit diesen lassen sich wesentliche Programmfunktionen aufrufen. Die weiteren Funktionen der Entwicklungsumgebung (Anlegen eines neuen Projekts, öffnen, speichern, drucken, usw.) werden über das Menü ausgeführt.

In der Mitte befindet sich der Hauptarbeitsbereich des Fensters. Hier werden alle Klassen eines Projekts und ihre Beziehungen (Vererbung und Benutzung) grafisch in einer UML-ähnlichen Form dargestellt. Durch Ziehen mit der Maus kann das Klassendiagramm gestaltet werden. Über das RechteMaus-Kontextmenü kann der Quelltext einer Klasse eingesehen und bearbeitet werden, sowie die Klasse compiliert werden.

Der untere Bereich heißt Objektleiste und dient zur Aufnahme der erzeugten Objekte. Eine der großen Stärken von BlueJ ist die Möglichkeit Objekte interaktiv zu erstellen. Mit dem Kontextmenü der Klasse können die Konstruktoren der Klasse direkt aufgerufen werden und dadurch Objekte erzeugt werden. Das Kontextmenü des Objekts erlaubt es die aktuellen Attributwerte zu inspizieren und Methoden des Objekts interaktiv aufzurufen.

Installation

Zuerst muss die Entwicklungsumgebung BlueJ vollständig auf dem Rechner installiert sein. Im Folgenden wird das Programm-Verzeichnis von BlueJ mit <BlueJ_Verzeichnis> bezeichnet (meist C:\Programme\BlueJ).

Kopieren Sie aus dem JavaKarol-Packet die Java-Bibliothek `javakarol.jar` in das Verzeichnis
< BlueJ_Verzeichnis>`\lib\userlib`
oder
Gehen Sie im BlueJ-Programm zu „Werkzeuge - Einstellungen“ - Reiter „Bibliotheken“ und fügen Sie dort die Datei `javakarol.jar` als Bibliotheks-Datei hinzu.

Jetzt stehen die Klassen der Java-Bibliothek für BlueJ-Programme zur Verfügung.

Karol-Projekt erstellen

0) Wenn Sie zum ersten Mal ein Karol-Projekt mit BlueJ erstellen, dann sollten Sie zuerst das Beispiel Laufen aus dem folgenden Kapitel nachvollziehen.

1) BlueJ Projekte sind Verzeichnisse – wie normale Java-Packages auch. Die Dateien, aus denen das Projekt besteht, sind in diesem Verzeichnis enthalten.
Als Vorlage für ein neues Karol-Projekt dient das Standardprojekt BlueKarol (siehe Kapitel Dateien). Diese Vorlage sollten Sie nicht verändern.
Fertigen Sie deshalb eine Kopie des Ordners
<JavaKarol_Verzeichnis>`\BlueJ\BlueKarol` an und speichern Sie diesen in Ihrem BlueJ-Arbeitsverzeichnis unter einem geeigneten Namen.

2) Starten Sie BlueJ. Mit „Projekt – Projekt öffnen …“ öffnen Sie das Projekt aus Schritt 2)

3) In Karol-Projekten sind die Aufgabenstellungen immer der Art, dass ein oder mehrere Roboter in einer festgelegten Welt eine Arbeit ausführen sollen (z. B. „sammle alle Ziegel“).
Das Vorlagenprojekt kann jetzt der Aufgabenstellung entsprechend angepasst werden. Hierzu können neue Klassen hinzugefügt werden oder die Klasse ROBOTER kann um neue Methoden ergänzt werden.
Es gibt verschiedene Möglichkeiten eine Karol-Aufgabenstellung mit einem BlueJ-Projekt zu bearbeiten. Hier einige:
   a. Interaktiv durch Erzeugen eines oder mehrerer Objekte der Klasse ROBOTER und mehrfachem, direktem Aufruf der Methoden dieser Objekte über das Kontextmenü.
   b. Die Klasse ROBOTER um neue Methoden ergänzen, von dieser geänderten Klasse ein Objekt erzeugen und bei diesem die neue Methode einmal interaktiv aufrufen.
   c. Eine neue Klasse ARBEIT anlegen („Neue Klasse“) und deren Konstruktor und Methoden in Java-Syntax festlegen („Bearbeiten“).
   In dieser Klasse ARBEIT werden die benötigten Objekte der Klassen WELT und ROBOTER als Referenzattribute definiert und im Konstruktor der Klasse ARBEIT erzeugt.
   In einer Methode Ausfuehren() der Klasse ARBEIT wird der eigentlich erforderliche

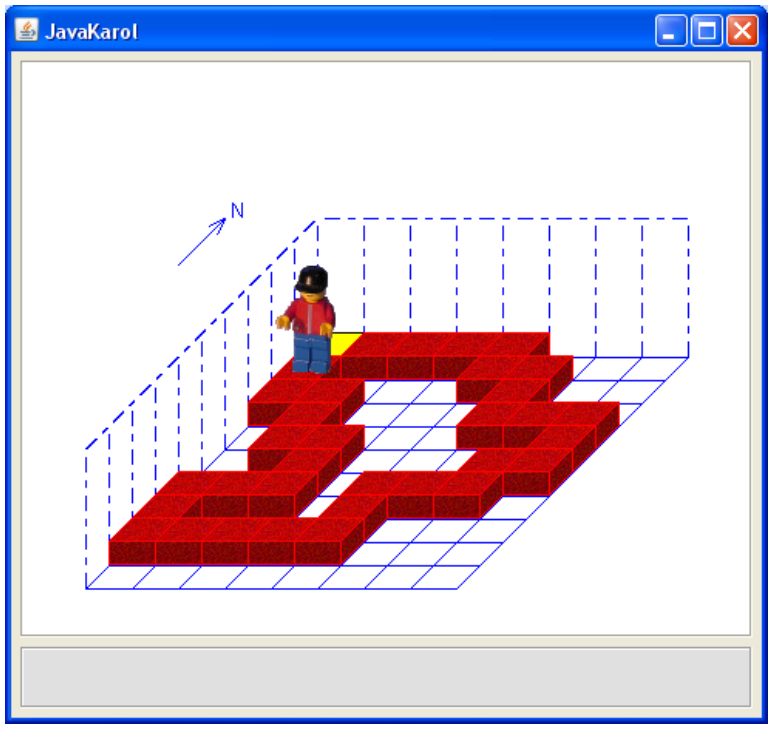
Ablauf in Java-Syntax unter Verwendung der Methoden eines Roboterobjekts festgelegt.
Von dieser Klasse ARBEIT erzeugt man dann ein Objekt (hierbei wird automatisch das JavaKarol-Fenster mit der Weltdarstellung geöffnet). Anschließend ruft man bei diesem Objekt die Methode Ausfuehren() auf.
Siehe Beispiel Laufen.

d. Von der Klasse ROBOTER werden Unterklassen definiert, die für die Aufgabenstellung spezialisierte Methoden bekommen und die restliche Funktionalität von der Klasse ROBOTER erben.
Wie bei c) wird eine Klasse ARBEIT angelegt, die jetzt Objekte dieser Unterklassen erzeugt und mit ihrer Methode Ausfuehren() für den gewünschten Ablauf sorgt.
Siehe Beispiel Grenze.

4) Egal auf welche Art die Karol-Aufgabenstellung bearbeitet wird: Es muss immer zuerst ein Objekt der Klasse WELT erzeugt werden und dann können Objekte der Klasse ROBOTER (oder Unterklassen) erstellt werden, die in diesem Weltobjekt „leben“.

5) Erzeugen eines Objekts der Klasse WELT.
Nach erfolgreicher Instanziierung öffnet sich automatisch ein JavaKarol-Fenster, in dem die Welt und soweit vorhanden Objekte der Klassen ZIEGEL, QUADER und MARKE dargestellt werden.
Für die ersten Beispiele empfiehlt es sich fertige Karolwelt-Definitionsdateien (aus dem Ordner <JavaKarol_Verzeichnis>`\Beispiele\Karolwelten`) zu verwenden.

6) Erzeugen eines Objekts der Klasse ROBOTER.
Nach erfolgreicher Instanziierung wird das Objekt als Roboterfigur in der Welt dargestellt.



7) Fehler beim Ablauf einer Methode.
Tritt während des Ablaufs einer Robotermethode ein Fehler auf (z. B. „Der Roboter stößt an die Wand“), so wird der Programmablauf unterbrochen. Im Textbereich des

JavaKarol-Fensters wird ein ausführlicher Text mit der Fehlerursache ausgegeben. BlueJ öffnet ein Fenster mit dem Quelltext und der Cursor steht an der verursachenden Programmzeile.

8) Benutzen von Karolwelt-Dateien.
In einer Karolwelt-Datei wird der Zustand einer Karolwelt und der darin enthaltenen Objekte abgespeichert und kann später wieder beim Erzeugen einer Welt verwendet werden.
Zu einem BlueJ-Projekt gehörige Karolwelten sollten immer im Projektverzeichnis gespeichert werden.

9) Wiederholter Ablauf.
Hat man die Karol-Aufgabenstellung nach dem Vorschlag von 3.c) oder 3.d) realisiert, so möchte man sicher die Methode Ausfuehren() mehrfach aufrufen. Dabei soll aber die Welt (und die darin enthaltenen Objekte) vorher wieder in den Ausgangszustand gebracht werden. Hierzu ergänzt man die Klasse ARBEIT um eine Methode Zuruecksetzen(), die wiederum die Methode WeltZurueckSetzen() des Weltobjekts aufruft. Durch Verwendung der Methode Zuruecksetzen() vor der Methode Ausfuehren() erreicht man das Gewünschte.

10) Arbeit mit dem Projekt beenden.
Dies ist möglich durch:
   a. Schließen des JavaKarol-Fensters (Schließfeld rechts oben). Dabei werden die erzeugten Objekte aus der Objektleiste entfernt, das Projekt an sich bleibt offen.
   b. Schließen des gesamten Projekts (Menü „Projekt schließen“). Dabei wird auch das JavaKarol-Fenster geschlossen.

Beispiel

Im Ordner <JavaKarol_Verzeichnis>`\BlueJ\Beispiele` befindet sich als exemplarisches BlueJ Karol-Projekt das Projekt Laufen. Öffnen Sie das Projekt über „Projekt – Projekt öffnen“.

Interaktive Steuerung

Sie können das Laufen des Roboters auf der Ziegelmauer (siehe Abbildung oben) interaktiv durchführen.

Erzeugen Sie ein Objekt der Klasse WELT über das Kontextmenü durch Auswahl des Konstruktors <new WELT(String weltdatei)>. Geben Sie dem Objekt den Instanznamen *welt* und als Parameter den Namen der passenden Karolwelt-Datei „Laufen.kdw“ (mit „“ Zeichen !). Sollte die Karolwelt-Datei nicht gefunden werden, so öffnet sich ein Dateiauswahl-Fenster in dem die Karolwelt-Datei *Laufen.kdw* gesucht werden kann (z. B. im Ordner <JavaKarol_Verzeichnis> `\BlueJ\Beispiele\Laufen`). Es öffnet sich das JavaKarol-Fenster mit der Darstellung der Welt.

Als nächstes erzeugen Sie ein Objekt der Klasse ROBOTER mit Hilfe des Konstruktors <new ROBOTER(WELT inWelt)>. Als Instanzname wählen Sie *karol* und als Referenzparameter geben Sie den Bezeichner der Karolwelt ein, in diesem Fall *welt* (ohne „ Zeichen!). Der Roboter wird im JavaKarol-Fenster angezeigt.

Jetzt können Sie den Roboter *karol* durch interaktiven Aufruf der Methoden (Kontextmenü des Objekts *karol*) bewegen.

Steuerung durch Programm

Im Projekt Laufen gibt es neben den Klassen WELT und ROBOTER auch noch eine Klasse ARBEIT. Betrachten Sie den Quelltext dieser Klasse (Doppelklick oder Kontextmenü Bearbeiten).

Sie sehen zwei Referenzattribute für die zu erzeugenden Objekte der Klasse WELT bzw. ROBOTER. Im Konstruktor ARBEIT() der Klasse ARBEIT werden die Objekte *meineWelt* und *karol* erzeugt.
Die Methode Ausfuehren() sorgt für den eigentlichen Ablauf. In der Methode Ausfuehren() sehen Sie, dass der Roboter *karol* durch Methodenaufrufe über die Ziegelmauer geleitet wird. Schließen Sie das Quellcode-Fenster.

Erzeugen Sie ein Objekt der Klasse ARBEIT durch den Konstruktor <new ARBEIT()>. Rufen Sie über das Kontextmenü des Objektes die Methode Ausfuehren() auf. Der Roboter *karol* bewegt sich auf der Ziegelmauer bis zur Markierung.

Bevor Sie die Methode Ausfuehren() erneut aufrufen sollten Sie die Methode ZurueckSetzen() verwenden um die Welt in den Ausgangszustand zu versetzen.

Ich wünsche Ihnen viel Spaß und einen großen Lernerfolg mit JavaKarol

U. Freiberger

Stand: 1. Novemver 2018