

## Ein spielerischer Einstieg in die Programmierung mit Kara und Java

von Raimond Reichert, Jürg Nievergelt und Werner Hartmann

**Wie bietet man Schüler/innen auf Stufe Gymnasium oder berufsbildender Schulen, die noch nie oder nur sehr wenig programmiert haben, einen spannenden Einblick und Einstieg in die Welt der Programmierung? Wie kann man langlebige, grundlegende Konzepte vermitteln, ohne sich mit der Komplexität professioneller Programmierungsumgebungen auseinandersetzen zu müssen? Kara, der programmierbare Marienkäfer, basiert auf dem einfachen Modell eines endlichen Automaten und bietet sich als graphische Einstiegsplattform für erste Programmierschritte an. LegoKara ist ein Lego Mindstorms-Roboter, der in der gleichen Programmierungsumgebung programmiert wird. JavaKara wiederum ist der virtuelle Marienkäfer, wird aber in Java programmiert und erlaubt so den nahtlosen Übergang von einer künstlichen Minisprache zu einer professionellen Sprache. Alle für den Unterricht benötigten Programme sind mit Beispielen und Unterrichtsmaterialien im Internet frei verfügbar unter [www.educeth.ch/informatik/karatojava](http://www.educeth.ch/informatik/karatojava).**

Professionelle Programmierungsumgebungen eignen sich im Unterricht nur schlecht für erste Schritte im Programmieren. Die Bedienung der Programmierungsumgebung, das Zusammenspiel der verschiedenen Komponenten wie Programmeditor, Compiler, Debugger oder Bibliotheken ist nicht auf Anhieb verständlich. Für Einsteiger ist auch die Syntax gängiger Programmiersprachen wie C/C++ oder Java alles andere als intuitiv. Und mit dem Einzug objektorientierter Programmierung stellt sich im Unterricht eine weitere Herausforderung: Die Kernideen objektorientierter Programmierung sind auf das Programmieren im Grossen ausgerichtet und liegen ausserhalb des Erfahrungsbereichs von Schüler/innen und Schülern. Kurz und gut: Professionelle Programmiersprachen und Programmierungsumgebungen sind für die ersten Programme im Unterricht wenig geeignet.

Wie soll der Einstieg ins Programmieren erfolgen? Es besteht die begründete Ansicht, dass den folgenden Anforderungen Rechnung zu tragen ist:

- Programmiersprache mit kleinem, rasch überblickbarem Sprachumfang
- Visuelle Darstellung des Programmablaufs
- Konkrete, anschauliche Problemstellungen

Schon seit langem versucht man diesen Anforderungen mit künstlichen Mini-Umgebungen gerecht zu werden. Eine solche künstliche Programmierwelt ist beispielsweise „Karel, the Robot“ [Pattis, 1981]. Die Idee, einen Roboter zu programmieren, der in einer Bildschirmwelt lebt, wurde immer wieder in der einen oder anderen Form aufgenommen. Wenn die Programme laufen, sehen die Schüler/innen visuell sofort, was sie programmiert haben und ob ihr Programm funktioniert. „Karel the Robot“ wird in einer Sprache programmiert, welche die wichtigsten Elemente einer imperativen Sprache enthält: einfache Anweisungen, if-Verzweigungen, while- und for-Schleifen sowie Prozeduren. Für Einsteiger eine bereits recht anspruchsvolle Hürde. Und oft erschweren Schwierigkeiten mit der Syntax der Sprache (fehlende Strichpunkte, Buchstabierung usw.) das Leben noch zusätzlich. Zudem bieten die meisten Mini-Umgebungen keinen nahtlosen Übergang zu einer produktiven Programmiersprache, sondern bilden eine eigene, in sich abgeschlossene Sprache.

## Kara – Eine Programmierumgebung mit endlichen Automaten

Bei den ersten Programmierschritten sollte nicht die relativ komplexe Syntax einer Sprache im Vordergrund stehen. Die Schüler/innen sollen sich zu Beginn auf das Wesentliche konzentrieren: auf das Denken in Abläufen, auf die Logik und Korrektheit ihrer Programme. Um dieses Ziel zu erreichen, kann man das Programmiermodell im Vergleich zu Umgebungen wie Karel the Robot noch stärker einschränken und statt einer allgemeinen Programmiersprache endliche Automaten benutzen. Diese Idee wird in [Nievergelt, 1999] vorgestellt und hat verschiedene Vorteile:

- Die Komplexität von endlichen Automaten ist deutlich geringer als die Komplexität allgemeiner Programmiersprachen. Das ist gerade für den Einstieg ins Programmieren ein nicht zu unterschätzender Vorteil!
- Der Speicher in Form von Zuständen erübrigt die Deklaration von Variablen und Datenstrukturen.
- Endliche Automaten begegnen uns im Alltag laufend und können etwa am Beispiel eines Geldautomaten anschaulich erklärt werden.

Ein endlicher Automat verknüpft den Speicher (seine Zustände) mit der Kontroll-Logik (die Zustandsübergänge). Will der Programmierer einen endlichen Automaten für ein gegebenes Problem erstellen, ist seine Arbeit einfacher, wenn er am Anfang alle Zustände des Automaten gedanklich erfasst, die bei der Lösung des Problems auftreten könnten. Er wird so ermutigt, die Aufgabe „ganzheitlich“ anzugehen. Dieses Vorgehen steht im Gegensatz zur „konventionellen“ Programmierung. Dort werden Anfänger oft verleitet, Programme zeilenweise „drauflos zu hacken“ und zusätzliche Variablen, Funktionen etc. schrittweise einzuführen. Bei diesem Vorgehen vermischt der Programmierer die Phasen der Problemanalyse und der eigentlichen Programmierung. Beim Erstellen der endlichen Automaten muss die Analyse **vor** der Programmierung erfolgen – was auch für Schüler/innen mit Programmiererfahrung eine lehrreiche Erfahrung ist.

Kara ist ein Marienkäfer [Reichert 2000], der in einer einfachen grafischen Welt auf dem Bildschirm lebt (Abbildung 1). Kara's Programme werden rein grafisch mit der Maus als endliche Automaten erstellt (Abbildung 2). Der Vorteil davon ist, dass jedes erstellte Programm sofort gestartet werden kann, weil keine Syntaxfehler wie „missing ;“ auftreten können. So wird die Einarbeitungszeit kurz gehalten. Wie sich bei mehreren Einführungskursen bestätigt hat, können Anfänger/innen typischerweise innerhalb von nur einer Stunde mit der Programmierumgebung von Kara so gut umgehen, dass sie sich auf das Lösen von anspruchsvolleren Aufgaben konzentrieren können.

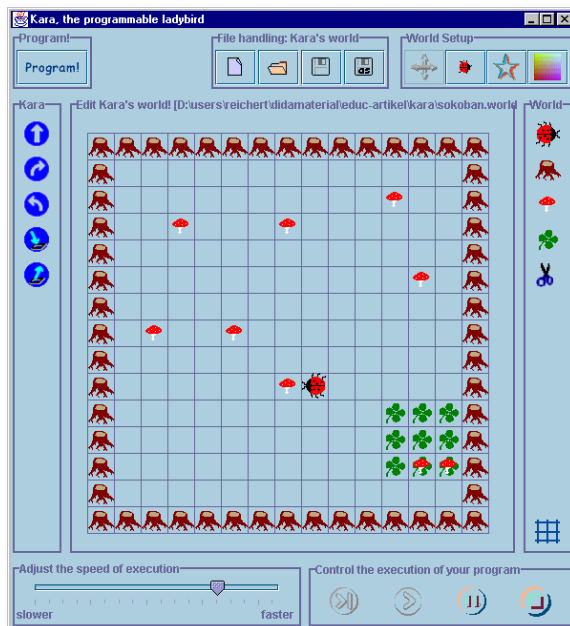


Abbildung 1: Kara und seine Welt

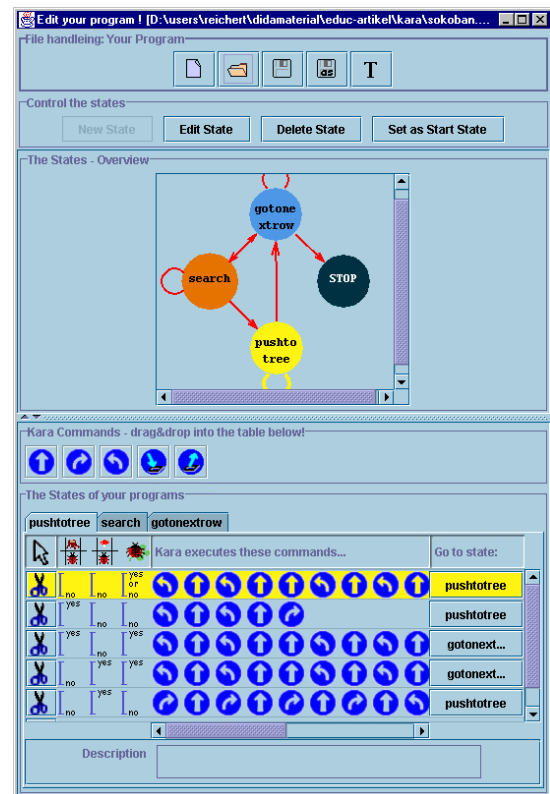







Abbildung 2: Die Programmierung von Kara

Kara lebt in einer rechteckigen Welt, die aus einzelnen Feldern besteht. Er kann sich nur von einem Feld zu einem benachbarten Feld bewegen. Auf den Feldern in dieser Welt gibt es (Abbildung 1):

- Unbewegliche Baumstümpfe: Kara kann diese Felder nicht betreten. Auf einem Feld mit einem Baumstumpf kann kein anderer Gegenstand liegen.
- Verschiebbare Pilze: Kara kann nicht auf einem Feld stehen, das von einem Pilz belegt ist. Läuft Kara in einen Pilz, so verschiebt er ihn geradeaus ins nächste Feld. Allerdings ist er zu schwach, um zwei Pilze gleichzeitig zu verschieben!
- Kleeblätter: Kara kann beliebig viele Kleeblätter aufnehmen. Er hat einen unerschöpflichen Vorrat an Blättern zum Ablegen. Kara kann auf einem Kleeblatt stehen, und ebenso können Pilze über Kleeblättern stehen.

Kara kennt nur wenige *Befehle*: einen Schritt vorwärts machen, auf dem aktuellen Feld um 90° nach links oder nach rechts drehen, ein Kleeblatt hinlegen oder aufnehmen. Damit Kara programmiert werden kann, verfügt er über *Sensoren*, mit deren Hilfe er seine unmittelbare Umgebung wahrnimmt. Kara kann bei jedem Zustandsübergang eine beliebige (konstante) Anzahl von Befehlen ausführen, in Abhängigkeit von den aktuellen Sensorenwerten:

-  Stehe ich vor einem Baumstumpf?
-  Ist links neben mir ein Baumstumpf?
-  Ist rechts neben mir ein Baumstumpf?
-  Stehe ich vor einem Pilz?
-  Stehe ich auf einem Kleeblatt?

Das ist alles, was Kara über die Welt weiss. Er weiss nicht, auf welchem Feld er steht oder in welche Himmelsrichtung er schaut. Natürlich könnte man Kara mit mehr Sensoren ausrüs-

ten. Aber die vorhandenen fünf Sensoren reichen aus, um viele interessante und anspruchsvolle Aufgaben zu lösen.

Da Kara seine Umgebung nicht nur wahrnimmt, sondern auch die Fähigkeit hat, sie zu verändern, kann er die Umgebung als externen Speicher benutzen. Wäre die Grösse der Welt unbeschränkt, so wäre Kara mit seiner Umgebung als Speicher eine universelle Turing Maschine, die beliebige berechenbare Funktionen berechnen könnte.

### Ein Programmbeispiel: Kara malt binäre Pascal-Dreiecke

Als Beispiel einer bereits anspruchsvollen Aufgabenstellung soll Kara ein binäres Pascal-Dreieck zeichnen. Jede Zahl wird „modulo 2“ abgebildet: eine gerade Zahl wird als freies Feld dargestellt, eine ungerade Zahl als Feld mit einem Kleeblatt. Der Einfachheit halber kippen wir das Dreieck in die linke obere Ecke der Welt. Abbildung 3 zeigt das entstehende Pascal-Dreieck.

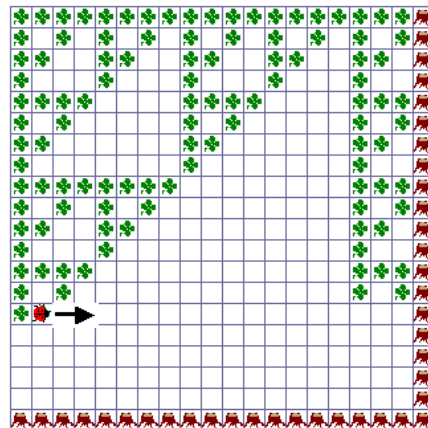


Abbildung 3: Pascal-Dreieck modulo 2

Die Berechnung der Werte im Pascal-Dreieck, das heisst, das Plazieren von Kleeblättern durch Kara, ist im Prinzip einfach: Kara berechnet das Pascal-Dreieck zeilenweise von links nach rechts. Erreicht Kara bei seiner Berechnung das Ende einer Zeile, markiert durch einen Baum, so läuft er im Zustand `next row` an den Anfang der nächsten Zeile.



Abbildung 4: Berechnung der Felder im Pascal-Dreieck

Wie berechnet Kara die Werte in den einzelnen Zellen des Pascal-Dreiecks? Kara kann über den Sensor „auf Kleeblatt?“ feststellen, ob er sich auf einem Feld mit einem Kleeblatt (also einer ungeraden Zahl) befindet oder nicht (das Feld **b** in Abbildung 4 links). Dann muss er noch wissen, ob sich auf dem Feld unmittelbar rechts hinter ihm (das Feld **a** in Abbildung 4; die eingekreisten Felder rechts) ein Kleeblatt befindet oder nicht. Diese Information wird dadurch abgespeichert, dass sich Kara in einem der beiden folgenden Zustände befindet:

- `carry0`: Auf dem Feld in Abbildung 4 diagonal links unterhalb von Kara (hinten rechts aus der Sicht von Kara) befindet sich kein Kleeblatt (also eine gerade Zahl)
- `carry1`: Auf dem Feld in Abbildung 4 diagonal links unterhalb von Kara (hinten rechts aus der Sicht von Kara) befindet sich ein Kleeblatt (also eine ungerade Zahl)

Befindet sich Kara im Zustand `carry0`, und steht er auf einem Feld ohne Kleeblatt, so bleibt das Feld rechts von Kara leer und Kara macht einen Schritt vorwärts. Falls er auf einem Feld mit einem Kleeblatt steht, so resultiert für das Feld rechts von Kara eine ungerade Zahl. Kara legt dort ein Kleeblatt hin und macht dann einen Schritt vorwärts. Entsprechend verhält sich Kara im Zustand `carry1`. Abbildung 5 zeigt das ganze Programm.

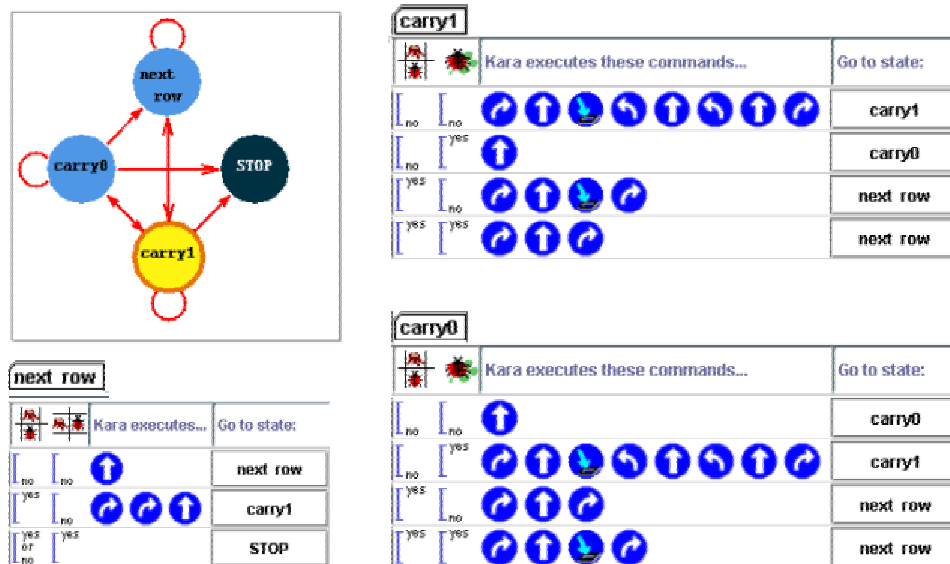


Abbildung 5: Das Programm für binäre Pascal-Dreiecke

Auch wenn Kara und die Welt, in der er lebt, sehr einfach gehalten sind, so reicht die Spannweite der Aufgaben dennoch von einfachen Einstiegsaufgaben bis hin zu anspruchsvollen Aufgaben. Kara bietet einen Einblick in die Gedankenwelt der Programmierung und hilft Berührungsängste abbauen. Insbesondere macht er Schüler/innen mit der Idee des Programmablaufs und der Logik vertraut und eignet sich somit hervorragend für Schüler/innen, die noch nie programmiert haben. Zudem lässt sich mit Kara das fundamentale Konzept der Korrektheitsüberlegungen zu Algorithmen anschaulich demonstrieren, da sich bei Automaten für wohldefinierte Aufgabenstellungen für jeden Zustand eine Invariante angeben lässt. Im obigen Beispiel sind die Invarianten der Zustände `carry0` und `carry1`, dass die Variable `a` in Abbildung 4 den Wert 0 (kein Kleeblatt rechts hinter Kara) beziehungsweise 1 (ein Kleeblatt rechts hinter Kara) hat.

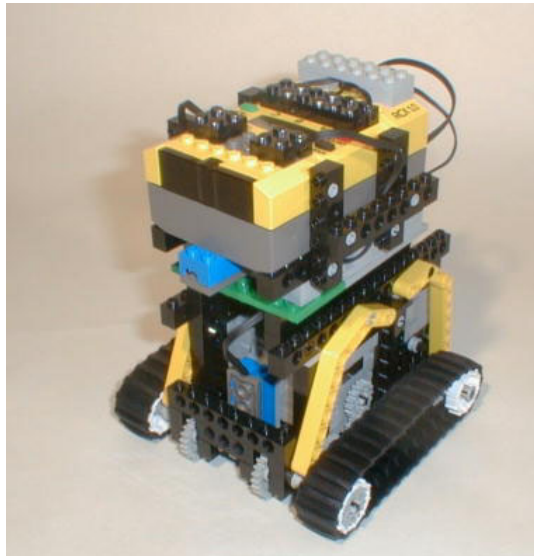
Allerdings bleibt Kara virtuell, auch wenn man sich einen echten Roboter plastisch vorstellen kann. Ein physischer Kara hingegen zeigt die Probleme, die beim Verlassen der wohldefinierten, digitalen Welt auftreten.

## LegoKara – Kara als Lego Mindstorms-Roboter

Mit dem Lego Mindstorms-Set (siehe <http://www.legomindstorms.com/>) können eigene Lego-Roboter gebastelt werden. Lego stellt auch eine Programmierungsumgebung zur Verfügung, um die Steuereinheit des Roboters zu programmieren. Im wesentlichen können aufgrund des Sensor-Inputs bestimmte Befehle ausgeführt werden. Ansonsten ist die Programmierungsumgebung beispielsweise in Bezug auf Kontrollstrukturen ziemlich eingeschränkt.

LegoKara [Meier, Reichert, Zürcher 2000] ist eine Realisierung von Kara als Lego-Mindstorms Roboter. Programmiert wird LegoKara in der grafischen Programmierungsumgebung Kara. Kenntnisse der Lego-Mindstorms Programmiersprache werden nicht benötigt. LegoKara stellt als real existierender Roboter (Abbildung 6) nur einen eingeschränkten Funktionsumfang von Kara zur Verfügung. LegoKara kann beispielsweise keine Kleeblätter aufnehmen.

Aufgrund einer detaillierten Bastelanleitung bastelt man sich in etwa zwei Stunden einen LegoKara. LegoKara erkennt, ob es zu seiner Linken oder Rechten oder vor ihm eine Wand hat und ob der Boden unter ihm hell oder dunkel ist. Einzig mit den Pilzen, den beweglichen Gegenständen in Kara's Welt, kann LegoKara nicht umgehen. Um Kara's Welt und Programmlogik einfach in die physische Welt abzubilden, macht LegoKara immer nur Schritte einer fixen Grösse und kennt nur 90°-Drehungen. Zwei klassische, für einen physischen Roboter wie LegoKara gut geeignete Aufgaben sind „Wand entlang laufen“ und „dunkle Spur am Boden verfolgen“. Allerdings dürfen Wand und Spur nur rechtwinklige Ecken haben.



**Abbildung 6: Der LegoKara-Roboter**

LegoKara ermöglicht es den Schüler/innen, ihre Programme an einem Roboter auszuprobieren. Aus der virtuellen Bewegung von Kara am Bildschirm wird eine Bewegung von LegoKara durch das physische Schulzimmer. Abgesehen davon, dass es Spass macht, mit physischen Robotern zu spielen, wird das Programmieren konkreter, weniger abstrakt. Zudem sieht man Probleme, die in der idealisierten, diskreten Welt von Kara am Bildschirm nicht auftreten – zum Beispiel wenn LegoKara meint, der Weg vor ihm sei frei, tatsächlich aber mit seinem in der Mitte angebrachten Sensor die Wand vor seinem Rad nicht wahrnimmt. Oder wie sehr eine 90°-Drehung auf den Raupen des Roboters von der Beschaffenheit des Bodens abhängt, wie genau die Parameter deshalb eingestellt werden müssen.

## **JavaKara – Die Schnittstelle zwischen Kara und Java**

Wie schaffen die Schüler/innen den Übergang aus einer künstlichen Miniwelt wie bei Karel the Robot zu einer Programmiersprache wie Java? Üblicherweise muss bei dem Übergang sowohl die vertraute Miniwelt als auch die dort geübte Programmiersprache aufgegeben werden.

JavaKara verfolgt einen anderen Ansatz und bietet einen sanfteren Einstieg in die Programmierung mit einer „realen“ Sprache wie Java. Kara macht die Schüler/innen mit der Miniwelt des Marienkäfers vertraut; mit JavaKara programmieren sie den gleichen Marienkäfer in der vertrauten Umgebung, aber sie programmieren ihn in Java. So können sie sich zu Beginn darauf konzentrieren, den Umgang mit dem verwendeten Programmeditor, dem Compiler `javac`, seinen nicht immer ganz einfach zu verstehenden Fehlermeldungen und dem Interpreter `java` mit bewusst einfachen Beispielen zu üben. Ist diese erste Hürde genommen, können die Schüler/innen Aufgaben steigender Schwierigkeit lösen, um so schrittweise die Sprachelemente von Java kennenzulernen.



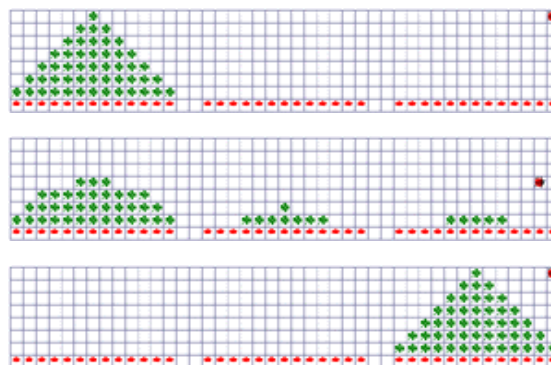
Um dabei am Anfang die Komplexität der objektorientierten Eigenschaften von Java auszu-  
blenden und die Konzentration auf die in jeder Programmiersprache vorhandenen Elemente  
wie Verzweigungen, Schleifen oder Basistypen und Arrays zu ermöglichen, kann eine  
Schablone (Abbildung 7) verwendet werden:

```
import roboapp.javakara.JavaKaraProgram;  
  
/* COMMANDS:  
 *   kara.move()           kara.turnRight()      kara.turnLeft()  
 *   kara.putLeaf()        kara.removeLeaf()  
 * SENSORS:  
 *   kara.treeFront()      kara.treeLeft()       kara.treeRight()  
 *   kara.mushroomFront()  kara.onLeaf()  
 */  
public class Template extends JavaKaraProgram {  
    // -----  
    // START DEFINE MY METHODS:  
    // . . .  
    // :END DEFINE MY METHODS  
    // -----  
  
    protected void myProgram() {  
        // -----  
        // START OF MY MAIN PROGRAM:  
        // . . .  
        // :END OF MY MAIN PROGRAM  
        // -----  
    }  
}
```

**Abbildung 7: Die Schablone für JavaKara-Programme**

Ein Programm für JavaKara muss von der Klasse JavaKaraProgram abgeleitet werden; das „Hauptprogramm“ muss in der Methode myProgram stehen. JavaKaraProgram stellt primär die Objekte kara und world zur Verfügung und bildet damit das Bindeglied zwischen eigenen Programmen und der Programmierungsumgebung JavaKara. Wie das im Detail funktioniert, ist auf den Webseiten [ Reichert 2000 ] erklärt. Wie die Kommentare in Abbildung 8 zeigen, sind die Schüler/innen von Anfang an auf ganz anschauliche Art und Weise mit Objekten konfrontiert: Kara selbst ist ein Objekt. Zu Beginn müssen die Schüler/innen nicht wissen, was das bedeutet; sie müssen einfach die für sie relevanten Methoden von Kara kennen wie zum Beispiel kara.move().

Abbildung 8 und 9 zeigen ein Programmbeispiel für das klassische Spiel der Türme von Hanoi. Aus Effizienzgründen wird direkt auf die Welt zugegriffen. Ein Programm von dieser Komplexität demonstriert einige grundlegende Konzepte wie zweidimensionale Arrays und Rekursion. Auch die „Berechnungen“ zur Platzierung der einzelnen Elemente der Türme, die hier als Kleeblätter-Stapel dargestellt sind, beinhalten Überlegungen, die in ähnlicher Form in vielen Programmen vorkommen.



**Abbildung 8: Türme von Hanoi zu**

```
import roboapp.javakara.JavaKaraProgram;

public class Hanoi extends JavaKaraProgram {
    final int towerDistance = 2;           // for the "layout" of the towers
    final int towerHeight    = 7;           // the numbers of disks
    final int towerWidth     = 2*towerHeight-1; // for the "layout" of the towers
    int[] middleOfTowers;                 // for the "layout" of the towers
    int[][] towers;                     // the disks of the towers
    int[] numbers;                       // number of disks of each tower

    void showTower (int[] disks, int n, int middle) {
        int x, y = world.getSizeY()-1;
        for (x = middle-towerWidth/2; x <= middle+towerWidth/2; x++)
            world.setMushroom (x, y, true);
        for (int j = 0; j < n; j++) {
            int width = disks[j]*2+1;
            for (x = middle-width/2; x <= middle+width/2; x++)
                world.setLeaf (x, y-1-j, true);
        }
    }

    void showAllTowers() {
        world.clearAll(); // remove all objects from world
        for (int i = 0; i < 3; i++)
            showTower (towers[i], numbers[i], middleOfTowers[i]);
        tools.checkState(); // wait a bit...
    }

    void moveDisk (int from, int to) {
        towers[to][numbers[to]] = towers[from][numbers[from]-1];
        numbers[from]--; numbers[to]++;
        showAllTowers();
    }

    void playHanoi (int numberOfDisks, int tower1, int tower2, int tower3) {
        if (numberOfDisks == 1)
            moveDisk (tower1, tower2);
        else {
            playHanoi (numberOfDisks-1, tower1, tower3, tower2);
            moveDisk (tower1, tower2);
            playHanoi (numberOfDisks-1, tower3, tower2, tower1);
        }
    }

    protected void myProgram() {
        middleOfTowers = new int[3];
        middleOfTowers[0] = towerWidth/2;
        middleOfTowers[1] = middleOfTowers[0]+towerWidth+towerDistance;
        middleOfTowers[2] = middleOfTowers[1]+towerWidth+towerDistance;
        towers = new int[3][];
        towers[0] = new int[towerHeight];
        towers[1] = new int[towerHeight];
        towers[2] = new int[towerHeight];
        numbers = new int[3];
        numbers[0] = towerHeight;
        numbers[1] = 0;
        numbers[2] = 0;
        for (int i = 0; i < towerHeight; i++)
            towers[0][i] = towerHeight-i-1;
        int totalWidth = towerWidth*3+towerDistance*2;
        int totalHeight = towerHeight+1;

        showAllTowers();
        playHanoi (numbers[0], 0, 2, 1);
    }
}
```

**Abbildung 9: JavaKara-Programm für Türme von Hanoi**

Auf der Website <http://www.educeth.ch/informatik/karatojava/javakara/> finden sich über 40 Beispiele zu JavaKara, gegliedert in 8 grössere Schritte, um so die Konzepte der Programmierung von Java schrittweise einzuführen. Die Beispiele decken primär die nicht-objektorientierten Eigenschaften von Java ab, bieten aber auch einen kleinen Einblick in die Objektorientierung. Haben die Schüler/innen die Konzepte hinter diesen Beispielen verstanden, so kennen sie Schleifen, Verzweigungen, Variablen, Methoden, Parameter, Arrays,



Klassen und Objekte und einiges mehr. Im Anschluss an diese Beispiele sollten sie in der Lage sein, sich mit „handelsüblichen“ Büchern zu Java mehr mit den objektorientierten Aspekten sowie mit den Klassenbibliotheken von Java beschäftigen zu können.

## Zusammenfassung

Professionelle Entwicklungsumgebungen eignen sich aufgrund ihrer Komplexität nicht für den Einstieg ins Programmieren im Unterricht. Da im Unterricht meist nur wenig Zeit zur Verfügung steht, eignen sich künstliche Mini-Umgebungen für erste Programmiererfahrungen. Mini-Welten haben aber meistens den Nachteil, dass sie keinen nahtlosen Übergang zu einer „echten“ Programmiersprache erlauben. Die Mini-Umgebung Kara, die Realisierung LegoKara und JavaKara schliessen genau diese Lücke: Von Kara für Schüler/innen ohne jede Programmiererfahrung bis hin zu JavaKara, der sich auch für Schüler/innen mit Erfahrung eignet. Kara, LegoKara und JavaKara sind einfach zu bedienen und einfach zu programmieren. Dennoch können mit ihnen fundamentale Ideen des Programmierens und der Informatik vermittelt werden.

Die Programme zu Kara, LegoKara und JavaKara sind im Internet frei verfügbar. Sie finden sich auf *EducETH*, dem Unterrichtsmaterialien-Server der Eidg. Technischen Hochschule Zürich unter <http://www.educeth.ch/informatik/karatojava/>. Nebst den Programmierungsumgebungen findet sich dort entsprechend der Philosophie von *EducETH* weiteres Material für den Unterrichts-Einsatz.

Raimond Reichert, Jürg Nievergelt, Werner Hartmann  
Departement Informatik  
Eidgenössische Technische Hochschule, ETH  
CH-8092 Zürich  
{reichert, nievergelt, hartmann}@inf.ethz.ch

## Literatur

Reichert, R.: Eine spielerische Einführung ins Programmieren: Von Kara nach Java. ETH Zürich, 2000. <http://educeth.ethz.ch/informatik/karatojava>

Meier, R.; Reichert, R. ; Zürcher, S.: LegoKara, die Lego-Mindstorms-Version von Kara. ETH Zürich, KS Baden, 2000. <http://educeth.ethz.ch/informatik/karatojava/legokara>

Pattis, R. E.: Karel the Robot: A Gentle Introduction to the Art of Programming. John Wiley & Sons, 1981.

Nievergelt, J.: „Roboter programmieren“ – ein Kinderspiel. Bewegt sich auch etwas in der Allgemeinbildung? Informatik Spektrum 22, Nr 5, 364-375, Oktober 1999.

Die Webseite zu dem Lego-Mindstorms-Bastelset von Lego.  
<http://www.legomindstorms.com/>