5.2 Programmierung mit Java

Java ist eine objektorientierte Programmiersprache. Sie ist plattformunabhängig, kann also auf unterschiedlichen Betriebssystemen operieren und gilt als eine sichere Sprache, da Java keine direkten Speicher- und Hardwarezugriffe unterstützt. Das macht sie insgesamt zu einer beliebten und weitverbreiteten Programmiersprache.

Im Verlauf dieses Abschnitts werden die Grundlagen der Sprache genauer erläutert. Viele der Elemente finden sich auch in anderen Sprachen wieder, daher wird Java oft als Einstieg in die Welt der Programmierung gewählt.

5.2.5 Ein bisschen Fachterminologie

Java gehört den höheren Programmiersprachen an. Das sind Programmiersprachen, die sich in Anbetracht ihrer Komplexität und Lesbarkeit deutlich von der Maschinensprache, der Sprache des Prozessors, unterscheiden. Eine höhere Programmiersprache wird in der Regel über mehrere Ebenen abstrahiert, sodass sie für uns Menschen verständlicher und besser lesbar ist. Damit ein Prozessor die Anweisungen eines Programms ausführen kann, muss also das Programm von einem Compiler über diese Abstraktionsebenen zurück in Maschinensprache übersetzt werden. Um ein erfolgreiches Übersetzen durch den Compiler zu gewährleisten, sind seitens des Programmierers verschiedene Regeln zu beachten.

In diesem Zusammenhang wird oft von Syntax und Semantik gesprochen. Die Syntax gibt an, wie der Code geschrieben werden muss, also an welche Regeln sich die Elemente des Codes zu halten haben. Die Syntax entspricht quasi der Grammatik der Programmiersprache. Beispielsweise werden alle Anweisungen jeweils mit einem Semikolon abgeschlossen. Die Semantik dagegen beschreibt, ob ein Codeelement inhaltlich sinnvoll ist. Eine Variablendeklaration kann beispielsweise syntaktisch richtig sein, jedoch im Programm nicht verwendet werden und ist dadurch nutzlos. Zu beachten ist, dass Menschen zwar dazu fähig sind, grammatikalisch falsche Aussagen zu interpretieren und, sofern sie einen semantischen Wert haben, angemessen zu handeln, Maschinen dies jedoch nicht können – was syntaktisch falsch ist, hat auch keine Semantik.

Im Laufe dieses Kapitels zur Programmierung mit Java werden wir immer wieder Bezug zum leJOS-Plug-in und der zugehörigen API nehmen. Was genau ist also ein Plug-in oder eine API?

Ein Plug-in ist eine Softwareerweiterung, die zusätzliche Funktionalitäten zu einem Anwendungsgebiet bereitstellt. So muss man nicht alles selbst programmieren, sondern kann auf vorgefertigte Funktionen zurückgreifen. Dabei kann es sich um einfache Module handeln oder um komplexe, umfangreiche Softwarebibliotheken. In den meisten Fällen werden die Plug-ins von den Nutzern

installiert. Damit eine Software auf diese Weise von Nutzern erweitert werden kann, definieren ihre Hersteller in der Regel eine sogenannte Programmierschnittstelle, eine API, über die Dritte überhaupt erst Zusatzsoftware dafür schreiben können. Gerade dann, wenn Plug-ins zum Schreiben von Quellcode gebraucht werden, besitzen auch diese eine API.

Schnittstellen allgemein schaffen Übergangspunkte, die der Kommunikation zwischen verschiedenen Komponenten dienen. Diese Komponenten können Softwarebibliotheken sein, Systeme, Hardware etc. Sprich, wo immer eine Komponente mit einem anderen kommunizieren will, sei es um eine Funktionalität bereitzustellen oder Daten zu übertragen und weiter zu verarbeiten, werden passende Schnittstellen (*engl. Interfaces*) benötigt. Sie legen fest, welche Funktionen und Eigenschaften eine Komponente nach außen hin repräsentiert und wie ein Datenaustausch erfolgen kann.

Eine API (*engl. für Application Programming Interface, kurz API*) ist nun im Speziellen eine »Schnittstelle zur Anwendungsprogrammierung«. Dadurch kann eine Software durch weitere Bibliotheken ergänzt werden, wie in unserem Fall mit Funktionen zur Programmierung mit EV3-Steinen von LEGO Mindstorms. Wichtig ist dabei eine ausführliche Dokumentation darüber, welche Funktion die Komponente enthält und wie diese Funktion im eigenen Code verwendet werden kann. Ziel einer API ist, dass der Programmierer dank der Dokumentation die Softwareerweiterungen nutzen kann, ohne sich die Implementierung der Funktionalitäten anschauen zu müssen.

Detailliertere Informationen und Beispiele zur Programmierung in Java findest du auf unserer Begleitwebsite. Dort gehen wir auch auf Themen ein, wie Variablen, Schleifen, bedingte Anweisungen und vieles mehr, was wir später verwenden werden. Wenn du noch tiefer in das Thema Java einsteigen möchtest, findest du in Kapitel 9.2 einige Buchtipps.

5.2.6 Auszug aus dem Programm des Roboters Vincent

Hier findest du einen Auszug aus dem Quellcode des Roboters Vincent. Dieses Java-Programm soll als Beispiel dienen, wie der Code aufgebaut sein könnte, und dir die mögliche Ablauflogik eines Roboters zeigen. Eine ausführliche Beschreibung von Vincent findest du in Kapitel 7.4.

Listing 5–1 // Auszug aus dem Programm des Roboters Vincent: Aufbau und Struktur eines Java-Programms

```
import lejos.hardware.port.MotorPort;
import lejos.hardware.port.SensorPort;

public class Vincent implements VincentInterface {

      public void run() {
         Robot.setLeftMotor(MotorPort.A);
         Robot.setRightMotor(MotorPort.B);
         Robot.setSensor(SensorPort.S1);
         Head.setMotor(MotorPort.C);
         Head.setSensor(SensorPort.S2);
         Robot.setSpeed(25);
         Robot.setTurnSpeed(60);

         // Wiederhole die Prozedur, solange die Bedingung (Vincent
         // hat das Ziel nicht erreicht) stimmt.
         while (!Robot.goalReached()) {
            // Drehe den Kopf von Vincent nach rechts. Ist dort eine
            // Wand?
            // Fall "keine Wand"
            // -----------------
            // Dreh Vincent nach rechts, dreh seinen Kopf nach vorne
            // und fahr nach vorne
            Head.lookRight();
            if (!Head.canSeeWall()) {
               Robot.turnRight();
               Head.lookForward();
               Robot.forward();
            } else {
               // Drehe den Kopf von Vincent nach vorne. Ist dort
               // eine Wand?
               // Fall "keine Wand"
               // -----------------
               // Fahr nach vorne
               Head.lookForward();
               if (!Head.canSeeWall()) {
                  Robot.forward();
               } else {
                  // Drehe den Kopf von Vincent nach links. Ist dort
                  // eine eine Wand?
                  // Fall "keine Wand"
                  // -----------------
```