Roberta®
Lernen mit Robotern

ROBERTA

EV3-Programmieren mit Java

Fraunhofer IAIS

Impressum

Ausgabe Februar 2015

Projektgruppe Beate Jost, Thorsten Leimbach, Maximilian Schöbel, Richard Erdmann, Daniel Pyka, Kostadin Cvejoski

Autor Maximilian Schöbel, Thorsten Leimbach, Beate Jost

Kontakt Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme IAIS
roberta@iais.fraunhofer.de
www.roberta-home.de

Umschlag-gestaltung Rebecca Zeller

ISBN ISBN (Print) 978-3-8396-0840-1

Information Bibliografische Information der Deutschen Nationalbibliothek: Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.de abrufbar.

Druck und Bindung Esser printSolutions GmbH, Bretten

Verlag © by FRAUNHOFER VERLAG, 2014
Fraunhofer-Informationszentrum Raum und Bau IRB
Postfach 800469, 70504 Stuttgart
Nobelstraße 12, 70569 Stuttgart
Telefon 0711 970-2500
Fax 0711 970-2508
verlag@fraunhofer.de
http://verlag.fraunhofer.de

Vorwort zu diesem Band

Im Jahr 2014 leben wir in einem Umfeld, das durch sich rasant entwickelnde Technologien geprägt ist. Computer sind aus unserem Alltag nicht mehr wegzudenken und die globale Wirtschaft wird durch Innovationen im Bereich der Informationstechnologie vorangetrieben.

Gemessen am technologischen Wandel ist Java, das 1995 entwickelt wurde, keine neue Programmiersprache. Damals steckte das Internet noch in den Kinderschuhen, Computer hielten Einzug in privaten Haushalten und in Büros und unterschiedliche Rechenplattformen kamen auf den Markt. Java wurde als leistungsstarkes, flexibles Werkzeug konzipiert, mit dem Programmierer Code nur einmal schreiben müssen, um ihn überall ausführen zu können. Java und die ihm zugrunde liegenden Konzepte sorgen dafür, dass Anwender konsistent und nahtlos mit Computern, zugehörigen Geräten und dem Internet interagieren können.

Haben Sie ein Smartphone oder einen Blu-Ray-Player? Spielen Sie Online-Games? Verlassen Sie sich auf Ihr Navigationssystem, wenn Sie mit dem Auto in einer fremden Gegend unterwegs sind? Surfen Sie im Web, nutzen Sie Online-Shopping oder haben Sie schon einmal Ihre elektronische Gesundheitsakte beim Arzt eingesehen?

Wenn ja, dann verwenden Sie Java.

Mit weltweit über neun Millionen Entwicklern ist Java mittlerweile allgegenwärtig. Als objektorientierte Programmiersprache wird Java auch in absehbarer Zeit das wichtigste Werkzeug bleiben, um bessere Websites, sichere Onlinetransaktionen und das Internet der Dinge zu ermöglichen.

Da Computer in unserem täglichen Leben überall anzutreffen sind, werden Kenntnisse in den Bereichen Informationstechnologie und Programmierung immer wichtiger, um aktiv an unserer globalen Gesellschaft teilzunehmen. Nicht zuletzt nimmt auch die Anzahl der Berufe, in denen IT-Fachwissen gefordert ist, immer mehr zu. Weltweit gibt es derzeit mehr Stellenangebote, die ein abgeschlossenes Informatikstudium

voraussetzen, als qualifizierte Bewerber. Zugleich eröffnet sich damit IT-Absolventen eine schier unbegrenzte Anzahl an Möglichkeiten.

Oracle Academy ist der Überzeugung, dass jedes Kind rund um den Globus Zugang zu einer guten Ausbildung im Bereich Informationstechnologie haben sollte. Schüler und Studenten müssen sich möglichst früh – idealerweise bereits im Grundschulalter – mit Informatik befassen und während ihrer gesamten schulischen und akademischen Laufbahn von Kursen und Lehrmaterialien begleitet werden, die Mathematik, den Umgang mit digitalen Technologien, Problemlösung und Programmierung beinhalten.

Aufgabe der Lehrkräfte ist es, entsprechende Lehrpläne zu erarbeiten und die Schüler und Studenten auf ihrem Weg unterstützend zu begleiten. Seit über 20 Jahren engagiert sich Oracle im Rahmen des Oracle Academy-Programms erfolgreich im Bereich Aus- und Weiterbildung. Gemeinsam mit Lehrkräften werden Ressourcen für Informatik- und Programmierkurse entwickelt und Lernenden auf der ganzen Welt zur Verfügung gestellt. Als Java-Steward bringt sich Oracle auch in verschiedene Java-Entwicklungsumgebungen (JDEs) ein, darunter Alice, Greenfoot, BlueJ, Eclipse und NetBeans. In unserer technologisierten Welt sind Java-Programmierkenntnisse nicht nur extrem wichtig, sondern auch enorm gefragt. Die Oracle Academy bietet umfangreiche Lehr- und Lernressourcen, um Schülern das geeignete Rüstzeug für ein erfolgreiches Studium und Berufsleben an die Hand zu geben.

Aus diesem Grund freuen wir uns besonders darüber, dass wir dieses Buch der Initiative „Roberta – Lernen mit Robotern“ des Fraunhofer-Instituts für Intelligente Analyse- und Informationssysteme IAIS unterstützen konnten. Die Roberta-Initiative liefert einen wertvollen Beitrag für die Wissensvermittlung. Seit über zehn Jahren bieten speziell ausgebildete und zertifizierte „Roberta-Teacher“ Hands-on-Roboterkurse an. Die Teilnehmer der Roberta-Kurse werden unter realistischen Bedingungen an die Entwicklung und Programmierung von mobilen Robotern herangeführt – eine faszinierende Erfahrung, die sie nachhaltig für Informationstechnologie begeistert und motiviert. Einige Schüler nehmen sogar an Roboter-Wettbewerben teil. Wir sind der Überzeugung, dass Jugendliche, die durch Robotertechnik oder auf andere Weise eine Leidenschaft für Computerwissenschaften entwickeln, im späteren Berufsleben erfolgreicher sein werden.

Durch die Zusammenarbeit mit der Roberta-Initiative und die Unterstützung des neuesten Roberta-Bands Programmieren mit Java leisten wir einen weiteren Beitrag zur Förderung der Aus- und Weiterbildung

im Bereich Informatik. Wir hoffen, durch die Bereitstellung von Lehr- und Lernmaterialien und -programmen eine durchaus komplexe Disziplin wie die Informatik für jedermann – insbesondere Mädchen und andere unterrepräsentierte Gruppen – zugänglich und interessant zu machen, um ihnen den Weg in eine bessere, erfolgreiche berufliche Zukunft zu ebnen.

Alison Derbenwick Miller
Vice President Oracle Academy

Geleitwort

Der Wirtschaftsstandort Deutschland ist angewiesen auf gut ausgebildete Nachwuchskräfte in naturwissenschaftlich-technischen Fächern. Ziel vieler Initiativen ist es deshalb, Kinder möglichst frühzeitig mit den »MINT«-Themen Mathematik, Informatik, Naturwissenschaften und Technik vertraut zu machen.

Das Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme IAIS hat bereits im Jahr 2002 die Initiative »Roberta® – Lernen mit Robotern« ins Leben gerufen: Roberta nutzt die Faszination von Robotern, um Schülerinnen und Schülern Naturwissenschaften, Technik und Informatik spannend und praxisnah zu vermitteln.

Es ist es eine spannende, wichtige, aber auch herausfordernde Aufgabe, diese Wissensgebiete an die nächsten Generationen zu vermitteln. Ihre erfolgreiche Bewältigung ist nicht einfach nur ein Teil des allgemeinen Bildungsauftrages – diese Aufgabe hat auch eine gesellschafts- und wirtschaftspolitische Bedeutung, die man nicht hoch genug einschätzen kann. In aktuellen Diskussionen um die Zukunftsaussichten unseres Landes wird immer wieder gefordert, dass wir (wieder) innovativer und mutiger werden, damit wir in der globalisierten Wirtschaft qualitativ hochwertige und global wettbewerbsfähige Produkte und Dienstleistungen anbieten können. Dazu brauchen wir jedoch junge Menschen, die unser Land durch eine erstklassige Ausbildung zu Ingenieurinnen und Ingenieuren und mit innovativen Ideen voran bringen.

Allerdings weisen Studien des Deutschen Instituts für Wirtschaftsforschung (DIW) hier auf einen enormen Mangel hin: Jährlich fehlen demnach unseren Wirtschaftsunternehmen rund 20.000 ausgebildete Ingenieurinnen und Ingenieure (Januar 2014). Vor allem die kleinen und mittelständischen Industrieunternehmen klagen über diesen Fachkräftemangel und halten ihn heute und in den nächsten Jahren für eine zentrale Herausforderung im Wettbewerb.

Wenn man die Entwicklung der Studierenden- und Absolventenzahlen in technischen Fächern an deutschen Hochschulen betrachtet, so stellt man fest, dass dieser Fachkräftemangel nur durch eine signifikante Steigerung der Anfängerzahlen nachhaltig ausgeglichen werden kann. Ei-

ne besondere Rolle kommt hierbei der Erhöhung des Anteils von Frauen zu, die weiterhin stark unterrepräsentiert sind. Wenn es gelingt, mehr Mädchen schon in der Schule für Technik, Mathematik und Naturwissenschaften zu begeistern, können neue Talente und Potentiale für die technischen Fächer erschlossen werden.

Wir freuen uns vor diesem Hintergrund besonders, dass sich das Roberta-Projekt mittlerweile zu einer erfolgreichen und umfangreichen Initiative entwickelt hat, zu der ein großes Netzwerk von Partnern entscheidend beiträgt. Jährlich besuchen über 35.000 Teilnehmerinnen und Teilnehmer, vorwiegend im Alter von 10 bis 16 Jahren, Roberta-Kurse und mehr als 94 Prozent äußern sich positiv und würden die Kurse weiterempfehlen. Ohne das persönliche Engagement der vielen Roberta-Begeisterten und ohne die Förderung durch unsere Partner aus öffentlicher Hand und Unternehmen wäre ein solcher Erfolg nicht möglich. Ihnen allen gilt deshalb unser besonderer Dank!

Wir hoffen, dass auch dieser Roberta-Band nützliches und spannendes Material bietet, um weitere Schülerinnen und Schüler für technische Themen zu begeistern.

Sankt Augustin, im September 2014

*P*rof. Dr. Stefan Wrobel
Institutsleiter Fraunhofer IAIS

Inhaltsverzeichnis

Kapitel 1

»Roberta – Lernen mit Robotern«

1.1 Einleitung

In Deutschland besteht seit Jahren ein Fachkräftemangel in technischen Berufen. Die Robotik bietet eine Chance diesem Mangel entgegenzuwirken, indem zukunftsweisende Techniken in die naturwissenschaftlich-technische Bildung integriert werden. Als eine Möglichkeit, dieser Herausforderung frühzeitig und angemessen zu begegnen, stellen wir das Roberta-Konzept vor und berichten über Erfahrungen mit diesem Konzept und Evaluierungsergebnisse.

Dabei erweist sich der Genderaspekt als wesentlich, um die bislang ungenutzten Potenziale des weiblichen Nachwuchses zu fördern. Um den Fachkräftemangel zu beheben und den Frauenanteil in technischen Berufen in Deutschland zu erhöhen, müssen mehr Mädchen bereits in der Schule für technische Fächer interessiert werden.

Roberta nutzt die Faszination von Robotern, um Schülerinnen und Schüler für Informatik und Technik zu begeistern und ihr Selbstvertrauen in ihre technischen Fähigkeiten zu stärken.

Roberta unterstützt Lehrerinnen, Lehrer und alle die, die Mädchen und Jungen für technische Berufe begeistern wollen, durch:

- ein Konzept für die Gestaltung von Roboter-Kursen, die insbesondere Mädchen ansprechen,
- Schulungen für Kursleiterinnen und -leiter, die die Qualität ihrer Kurse sichern,
- umfassende Lehrmaterialien, mit denen geschulte Roberta-Kursleiterinnen und -leiter Roboter-Kurse durchführen können,
- ein bundesweites Netzwerk von RobertaRegioZentren, in denen RobertaKursleiterinnen und -leiter ortsnah Unterstützung finden,

- das Roberta-Portal als ständigen Begleiter der zertifizierten Roberta-Lehrkräfte,
  - auf dem sie alle wichtigen und aktuellen Informationen zu Roberta finden,
  - über das sie mit anderen Lehrkräften in Kontakt kommen und Erfahrungen austauschen können,
  - auf dem sie ihre Kurse dokumentieren und andere Veranstaltungen bekanntgeben können,
- die Zusammenarbeit mit RobertaCompetenceCenter (RCC), die sich an der Weiterentwicklung des Roberta-Konzeptes und der Roberta-Inhalte beteiligen.

Herausforderung und Statistik

In einem rohstoffarmen Land wie Deutschland ist zum Fortbestand eines qualifizierten Standards eine auf hohem Niveau ausgebildete Bevölkerung unabdingbar.

Zukunftsträchtige Bereiche, die Arbeitsplätze bereitstellen, basieren oft auf mathematisch-naturwissenschaftlicher und informatischer Ausbildung. Dies zeigen deutlich die Zahlen des Arbeitsmarktes der letzten Jahre. Die meisten Stellenangebote gibt es für Fachrichtungen des Maschinenbau- und Elektroingenieurwesens. Insgesamt sehen die Zahlen für Akademikerinnen und Akademiker deutlich positiver aus als für Nicht-Akademikerinnen und -Akademiker.

	Arbeitslosigkeit		
Bereich	2007	2012	2013
AkademikerInnen	3,7%	2,4%	k.A.
Bevölkerung	9,1%	6,8%	ca. 6,9%

Tabelle 1.1.: Verteilung der Arbeitslosigkeit (Bundesagentur für Arbeit 2014)

Namhafte Arbeitsmarktexperten prognostizieren einen anhaltenden erheblichen Mangel an Ingenieurinnen und Ingenieuren in Deutschland. Der jährliche Fehlbedarf liegt zurzeit bei 20.000 (Deutsches Institut für Wirtschaftsforschung 2014) bis 40.000 (Kayser und Koppel 2013). Wenn man bedenkt, dass etwa 100.000 Ingenieurstellen (Kayser und Koppel 2013) nicht besetzt sind (leicht fallende Tendenz 2013) und nur ca. 41.000 Studenteninnen und Studenten in Deutschland jährlich ihr Studium in den Ingenieurwissenschaften mit einem Bachelorab-

schluss abschließen (DESTATIS 2014), wird erkennbar, dass ein zusätzlicher Ausbildungsbedarf besteht[1].

Um den Fachkräftemangel in technischen Berufen zu beheben, müssen sehr viel mehr junge Menschen als bisher zu einer technikorientierten Ausbildung ermuntert werden. Sie müssen möglichst frühzeitig motiviert werden, damit sie bereits in den Schulen ein Interesse an Technik und Informatik entwickeln. Hierbei liegt ein großes, bisher weitgehend vernachlässigtes Potential bei den Mädchen. Ihr Interesse für technische Fächer und Berufe ist noch geringer als bei Jungen. Dass einige Maßnahmen bereits gefruchtet haben, erkennt man daran, dass der Anteil von Frauen bei Studienanfängern im Studienfach Informatik im Jahr 2006 bei 16,9% lag, 2013 bereits bei 22,5%. Ähnliches ist in der Elektrotechnik zu beobachten. Hier lag der Anteil 2006 bei 9,6% und 2012 bei 12,7% (Kompetenzzentrum Technik - Diversity - Chancengleichheit e.V. 2014).

Chancen

Seit einiger Zeit berichten die Medien vermehrt über die Robotik. Die Faszination, die Roboter insbesondere auf Jugendliche ausüben, lässt sich nutzen, um sie an komplexe Technologien heranzuführen, Hemmschwellen zu überwinden und Skepsis oder gar Ablehnung gegenüber Technik abzubauen. Durch die eigene Gestaltung von Robotern kann Freude und Interesse an Technik geweckt und die Lernbereitschaft erhöht werden (Bredenfeld und Leimbach 2010). Roboter sind gut geeignet um Wissen über das Funktionieren und Entwickeln technischer Systeme zu vermitteln und diese »begreifbar« zu machen. Grundlagen der Informatik sowie Zusammenhänge mit anderen technischen Disziplinen können spielerisch erklärt werden. Roboter sind sowohl für eine theoretische Grundbildung, als auch für die praktische Aus- und Weiterbildung in vielen Bereichen der Technik geeignet. Anwendungsbezüge lassen sich in umfangreichem Maß herstellen.

Die Arbeit mit Robotern spricht auch Mädchen und Frauen an. Das gilt insbesondere für Roboter mit Service- und Rettungsaufgaben sowie für Roboter, die naturwissenschaftliche oder biologische Phänomene simulieren. Dennoch trauen sich Mädchen oft nicht zu, die notwendigen Voraussetzungen für ein mathematisch-naturwissenschaftliches Studium oder gar für ein Berufsziel in Richtung Mechatronik zu erfüllen. Hier setzt Roberta an. Roberta begleitet insbesondere Mädchen in die faszinierende Welt der Roboter und zeigt ihnen, dass Informatik, Naturwissenschaften und Technik spannend sind. Das Selbstvertrauen

1 Eine gute Übersicht bietet das Online-Portal Statista unter: http://de.statista.com/statistik/daten/studie/247927/umfrage/absolventen-in-der-faechergruppe-ingenieurwissenschaften-an-deutschen-hochschulen/

der Mädchen wird durch die eigene Gestaltung der Roboter gestärkt. Gleichzeitig werden sie mit diesem »hands-on« Lern-Ansatz (Druin und Hendler 2000; Papert 1980) in die entsprechenden Wissensbereiche eingeführt.

Die Erfahrungen mit Roberta in den vergangenen 12 Jahren haben gezeigt, dass das Konzept erfolgreich ist, sich in ganz Deutschland immer weiter verbreitet und darüber hinaus Akzeptanz und begeisterte Zustimmung findet. Es kann zügig an technische Weiterentwicklungen angepasst werden und ist offen für das Hinzunehmen neuer technischer Produkte, wie zum Beispiel mobile Endgeräte (Smartphones).

1.2 Roboter in der Bildung

Roboterbaukästen

Roboterbaukästen wie LEGO® MINDSTORMS® EV3 und NXT (früher auch das RIS Robotic Invention System) erfreuen sich großer Beliebtheit. Sie bestehen aus einer Menge von mechanischen und elektronischen Bauteilen, die aufeinander abgestimmt sind und das Konstruieren wie Programmieren unterschiedlicher Robotertypen ermöglichen. Sie erlauben, komplexe Systeme – einzelne Roboter, aber auch Gruppen miteinander kommunizierender Roboter – mit einfachen Mitteln zu bauen und zu programmieren.

Erfolgsfaktoren

Der Erfolg von LEGO® Roboterbaukästen in der Ausbildung wird wesentlich durch folgende Faktoren bestimmt:

- Roboter sind konkrete, »anfassbare« Gegenstände (»hands-on«). Formeln bekommen einen Sinn.
- Roboterbaukästen ermöglichen einen einfachen Einstieg in die Programmierung, erste Erfolge sind unmittelbar sichtbar.
- LEGO® MINDSTORMS® ermöglicht einen einfachen Einstieg in die Programmierung, sodass kurzfristig Erfolgserlebnisse einsetzen.
- Ausgehend von einfachen Aufgaben lassen sich zunehmend komplexere Aufgaben lösen. Dies gilt insbesondere für die Programmierung.
- Das Entwickeln der Roboter erfordert einen vollständigen Systementwicklungsprozess, der vom Entwurf über die Konstruktion und die Programmierung bis zum Test reicht und viele Disziplinen einschließt, da Hardware, Software, Elektronik, Elektrik und Mechanik zusammenpassen müssen.

- Es gibt Programmiersprachen für unterschiedliche Anforderungen. So gibt es für fast alle Baukästen grafische Programmierumgebungen für den Einstieg, die auch für Computer-Laien keine Hemmschwelle darstellen, andererseits aber auch mächtigere Sprachen für diejenigen, die schwierigere Aufgaben lösen wollen. Auch praxisrelevante Programmiersprachen (wie Java) sind verfügbar.
- Das LEGO® MINDSTORMS® -System kann von der Grundschule bis zur Hochschule eingesetzt werden.
- LEGO® setzt auf einen offenen Ansatz (open-source). Dies ermöglicht die leichte Kombination mit anderen Soft- und Hardwaresystemen. Beispielsweise können Smartphones mit LEGO® MINDSTORMS® genutzt werden.

Roberta – Lernen mit Robotern

Roboter bieten einen auf Technik bezogenen Zugang zur Informationstechnik. Softwaretechnik und Rechnerarchitektur können ebenso angesprochen werden wie Arbeitsorganisation, Projektplanung und wichtige Grundlagen der Mathematik oder Physik. Dabei werden fachliche Kenntnisse und Methodenwissen erworben. Nicht-fachliche Kompetenzen, sogenannte Soft Skills wie etwa Teamfähigkeit, werden gefördert und Interagieren, Kommunizieren, Präsentieren und Dokumentieren geübt.

Gender

Unterschiede zwischen den Geschlechtern werden vielfach nicht explizit berücksichtigt, sondern eher als gegeben hingenommen. Klassische Schulforschungsansätze gehen häufig von einer Geschlechterdifferenz aus. So wird etwa Jungen per se eher naturwissenschaftliche, Mädchen dagegen eher sprachliche Kompetenz zugeordnet. Die Erfahrungen in der Realität scheinen diese Annahme im Wesentlichen zu bestätigen.

Im Englischen gibt es für das biologische und das soziale Geschlecht unterschiedliche Begriffe:

- »Sex« bezeichnet das biologische,
- »Gender« das sozial und kulturell geprägte Geschlecht.

Da es eine solche Unterscheidung im Deutschen nicht gibt, werden zur Differenzierung manchmal die englischen Begriffe verwendet. Wird die Annahme einer naturgegebenen Geschlechterdifferenz hinterfragt – wie das die Geschlechterforschung leistet – kann dies zu erstaunlichen Ergebnissen führen. Zwei nachdenklich stimmende Beispiele kommen aus der Geschlechterforschung im Unterrichtsfach Physik:

- In einer Untersuchung wurde festgestellt, dass Mädchen bei der Bewertung einer schriftlichen Physik-Arbeit bessere Chancen haben, die Bestnote zu erreichen, wenn nicht zu erkennen ist, dass sie von einem Mädchen stammt, wenn also das Bewertungsverfahren anonymisiert ist (Frank 1995b) (Frank 1995a). Ähnliche Untersuchungsergebnisse gibt es bezogen auf Jungen und Sprachen.
- Eine andere Untersuchung bezieht sich auf das Vermitteln des Prinzips einer Pumpe im Physikunterricht (Häußler und Hoffmann 1998). Mädchen zeigen großes Interesse, wenn die Pumpe als künstliches Herz eingeführt wird. Dagegen ist ihr Interesse für eine erdölfördernde Pumpe geringer. Jungen interessieren sich für beide Pumpenarten in gleicher Weise und sind sogar bereit, eine rein wissenschaftliche, abstrakte (formelorientierte) Betrachtung zu akzeptieren.

Für die Mädchen kommt es also auch auf die »Verpackung« an. Bei den Jungen sind in den vergangenen Jahren zunehmend Überforderungsprobleme erkannt worden, die auf »Leistungsdruck« zurückzuführen sind. Auch die Jungen brauchen daher eine besondere Förderung durch die Lehrkräfte, die ihnen den Druck nimmt, aber nicht das Vertrauen in ihre Leistungsfähigkeit.

Roberta setzt an diesen Erkenntnissen an und ist beispielhaft dafür, wie Mädchen und Jungen gleichermaßen für Technik begeistert werden können.

Diversity

Im Sprachgebrauch von Wirtschaftsunternehmen taucht der Begriff »Diversity« (Vielfalt) häufig auf.

Diversity steht für ein Konzept, das die unterschiedlichen Ideen und Perspektiven einer vielfältig zusammengesetzten Belegschaft gezielt nutzt, um die Bedürfnisse der Kundschaft zu erfassen. Diversity bezeichnet demgemäß unterschiedliche Merkmale: neben dem Geschlecht unter anderem die familiäre Situation, das Bildungsniveau der Eltern, die ethnische Herkunft und die soziale Einbindung. Im Diversity-Ansatz bedeutet Geschlechtergerechtigkeit zum Beispiel, Sicherheitsgurte zu entwickeln, die eine Schwangerschaft bei einem Unfall nicht gefährden oder Mobiltelefone, die auch höhere weibliche Stimmlagen sauber übertragen.

Bei Lernszenarien kommt es folglich darauf an, die Freude an Naturwissenschaft und Technik sowohl Mädchen als auch Jungen zu vermitteln. Durch frühzeitiges Berücksichtigen von Genderaspekten – indem

einerseits Lehr- und Lernmaterialien und andererseits Verhaltensweisen und didaktische Maßnahmen der Lehrkräfte an den Bedürfnissen beider Geschlechter orientiert sind – wird die Qualität der Ausbildung für beide Geschlechter verbessert. Diversity-Merkmale werden von Roberta im Genderbereich angesprochen, in die Themen und Experimente einbezogen (keine Diskriminierung) und durch spezielle Kurse (etwa für Kinder mit Migrationshintergrund, die gemeinsam mit ihren Müttern teilnehmen) berücksichtigt.

1.3 Das Konzept »Roberta«

Zum Roberta-Konzept gehören:

Netzwerk
: Roberta-Teacher finden Unterstützung bei den RobertaRegioZentren.

Roberta-Coaches
: Nur von Fraunhofer IAIS akkreditierte Personen (Roberta-Coaches) dürfen Roberta-Schulungen (Roberta-Teacher-Trainings) durchführen. Roberta-Coaches verfügen über eine ausgewiesene Expertise in den Bereichen Didaktik und Technik. Zudem besitzen sie langjährige Erfahrung als Roberta-Teacher.

Schulungen
: Roberta-Teacher-Trainings umfassen die Einführung in die Handhabung der Roberta-Baukastensysteme und vermitteln den sicheren Umgang mit der Hard- und Software der Roboter.

Zertifizierte Kurse
: Nicht jeder Roboter-Kurs ist ein Roberta-Kurs. Nur geschulte und zertifizierte Roberta-Teacher dürfen Roberta-Kurse durchführen.

Lehrmaterial
: Lehrmaterialien sind in der Roberta-Reihe zusammengestellt. Zusätzlich können zertifizierte und im Roberta-Portal registrierte Roberta-Teacher ergänzende Roberta-Materialien kostenlos herunterladen.

Roberta-Portal
: Für alle an Roberta Interessierten, mit speziellen Bereichen für registrierte Roberta-Lehrkräfte (Roberta-Teacher).

Diese Bestandteile werden im Folgenden erläutert.

Zudem werden verschiedene Baukästen vorgestellt, wobei das Roberta-Konzept bisher mit dem LEGO® MINDSTORMS® Robotics Invention System, dem LEGO® MINDSTORMS® EDUCATION NXT System und dem LEGO® MINDSTORMS® Education EV3 System umgesetzt wurde.

Netzwerk

Das Roberta-Netzwerk dient sowohl der Verbreitung der Kurse als auch der Unterstützung der Roberta-Teacher (KursleiterInnen). Es soll den regionalen wie auch den überregionalen Erfahrungsaustausch fördern und langfristig dazu führen, Einzelaktivitäten zu bündeln. Die RobertaRegioZentren koordinieren die Kurse in ihrer Region, sie schulen und betreuen die KursleiterInnen (z. B. Lehrkräfte, Erzieherinnen/Erzieher, Schülerinnen/Schüler und Studierende) oder verleihen bei Bedarf Baukästen zum Durchführen der Kurse.

Eine aktuelle Liste der derzeit bestehenden RobertaRegioZentren (RRZ) finden Sie im Internet über die Adresse: www.roberta-home.de/de/netzwerk/deutschland

Schulungen

Die Schulungen (Teach-the-Teacher) vermitteln den künftigen Roberta-KursleiterInnen das Roberta-Konzept sowie den Umgang mit dem Roboterbaukastensystem und den Roberta-Materialien. Um die angehenden Roberta-Teacher für Genderaspekte zu sensibilisieren, werden bei der Schulung geschlechtsspezifische Unterschiede im Lehrverhalten von Kursleiterinnen und -leitern sowie im Lernverhalten von Kursteilnehmerinnen und -teilnehmern thematisiert und mögliche Verhaltensmuster angesprochen.

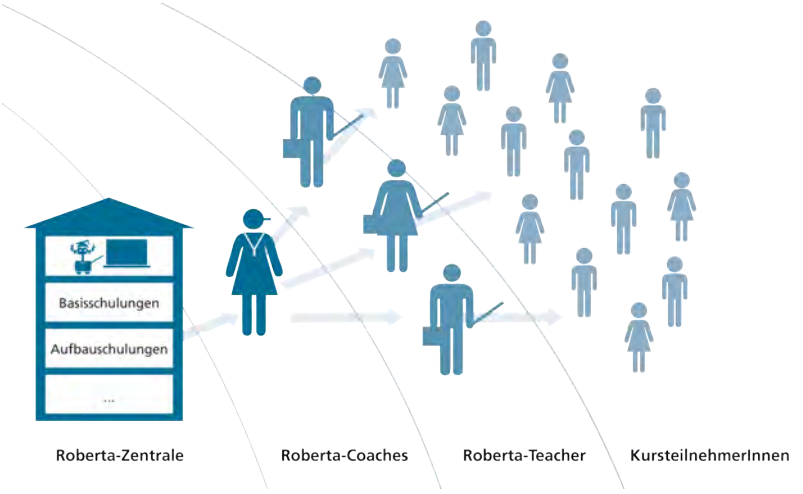


Abbildung 1.1.: MultiplikatorInnen-Schulung innerhalb des Roberta-Konzepts

In den Schulungen erfahren die zukünftigen Roberta-Teacher, welche Vorbereitungen notwendig sind, um einen Roberta-Kurs durchzuführen und wie sie die Roberta-Materialien nutzen können. Sie lernen die Roboterbaukästen kennen und üben das Konstruieren und Programmieren der Roboter. Sie sollen anschließend in der Lage sein, einen Roberta-Kurs selbstständig durchzuführen und anhand der Materialien eigenständig weiterzuarbeiten.

Die Schulungen werden vom Fraunhofer IAIS sowie von akkreditieren Roberta-Coaches durchgeführt. Sie dauern zwischen 8 und 12 (Zeit-) Stunden und sind wegen der Fülle des zu vermittelnden Stoffes meist auf zwei Tage verteilt. Die Schulungen sind modular aufgebaut. Derzeit kann zwischen folgende Schulungen gewählt werden:

- Roberta-Basisschulung (Zertifizierung zum Roberta-Teacher)
- Roberta eXperts Schulung (für fortgeschrittene Roberta-Teacher)
- Roberta-Schulung Programmieren mit Java
- Roberta-Schulung Programmieren mit dem Android App Inventor
- Roberta-Schulung Programmieren mit LabVIEW
- Roberta-Schulung Programmieren mit NXC

Um die hohe Qualität der Schulungen langfristig gewährleisten zu können, werden alle Roberta-Schulungen (seit 2008) evaluiert. Der dafür entwickelte Fragebogen umfasst zu bewertende Aussagen wie etwa:

- »Die Inhalte der Schulung helfen mir bei der Planung, Durchführung und Weiterentwicklung meiner Roboterkurse.«
- »Die Arbeitsmethoden sind im Hinblick auf das Ziel der Schulung angemessen.«
- »Die Schulung war lebendig und interessant gestaltet.«

Insgesamt beurteilen 97,5% aller in den Roberta-Schulungen befragten Personen die Schulung für gut bis sehr gut!

Ergebnisse Roberta-Teacher-Training				
Sehr gut	Gut	Zufrieden	Nicht Zufrieden	Insgesamt
67,94%	29,56%	2,50%	0%	100%
462	201	17	0	680

Tabelle 1.2.: Evaluationsergebnisse des Roberta-Teacher-Trainings (2008-2013)

In den letzten Jahren wurden über 1000 Personen als Roberta-Teacher zertifiziert. Die nachfolgende Grafik zeigt, wie sich die Anzahl der Teilnehmerinnen und Teilnehmer seit 2008 entwickelt hat.

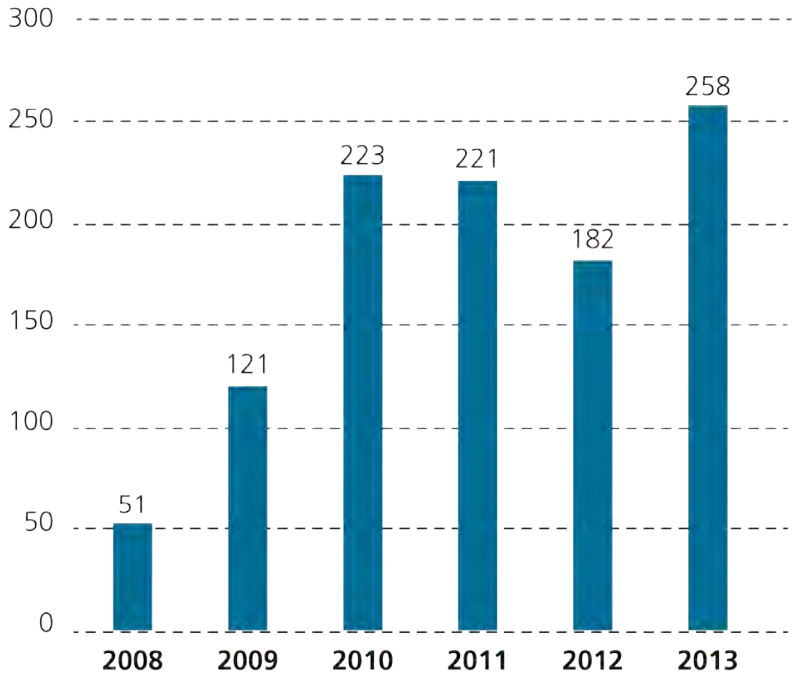


Abbildung 1.2.: Zertifizierte Roberta-Teacher 2008-2013

Seit dem Ablauf der Förderung (von BMBF und EU) im Jahr 2008 werden die Roberta-Schulungen seit 2008 kostenpflichtig angeboten. 2010 wurden sie in das Angebot der Fraunhofer Academy aufgenommen.

Kurse

Roberta-Kurse unterscheiden sich von vielen der heute angebotenen Roboter-Kurse durch die folgenden Anforderungen:

- Die Kursleitung (Roberta-Teacher) hat erfolgreich an einer Roberta-Schulung teilgenommen.
- Die Kursleitung ist sensibel für die eigenen geschlechtsspezifischen Verhaltensweisen.
- Die Kursleitung geht auf unterschiedliche Lernweisen von Mädchen und Jungen ein.

- Die Aufgaben sind in Themenstellungen eingebunden, die für Mädchen und Jungen interessant sind.
- Die Kursleitung ist aktiver Teil eines europaweiten Roberta-Netzwerks.

Die Gendersensitivität der Kursleitung trägt dazu bei, dass die naturwissenschaftlich-technischen Interessen beider Geschlechter gefördert werden. Roberta-Teacher können sich über das Portal austauschen und so das Roberta-Konzept mit weiteren Beispielen aus der Praxis sinnvoll ergänzen.

Lehrmaterialien

Die Roberta-Lern- und Lehrmaterialien setzen sich im Wesentlichen aus der Roberta-Reihe und den im Roberta-Portal verfügbaren Online-Materialien zusammen. Die Lern- und Lehrmaterialien werden ständig erweitert und an Entwicklungen des Roberta-Konzepts angepasst:

- Anregungen zum Einsatz weiterer Programmiersprachen bietet beispielsweise der Roberta-Band »Programmieren mit Java«.
- Anregungen für weitere Themen in langen Kursen finden sich im Roberta Band »Experimentieren mit EV3« bzw. »Experimentieren mit NXT«.

Roberta-Box

Mit der Roberta-Box wird eine Arbeitsgruppe von zwei bis drei SchülerInnen grundlegend mit Materialien zum Bauen und Programmieren von Robotern nach dem Roberta-Konzept ausgestattet. Die Roberta-Box ist wahlweise mit oder ohne Software-Lizenz erhältlich.

Abbildung 1.3.: Die Roberta-Box

Die Roberta-Box enthält folgende Elemente:

- LEGO® MINDSTORMS® Education EV3-Basis-Set
- EV3-Software (optional)
- Verzierungsteile
- LEGO® Transformator
- Roberta-Band: EV3-Geschöpfe aus einem Baukasten
- Maßband
- Roberta-Testparcours (Folie)
- Farbige Roberta-Klebefolien
- Blaue Kunststoffbox mit Deckel

Roberta-Portal

Das neue Internetportal www.roberta-home.de bietet ganz im Sinne eines sozialen Netzwerkes Funktionen zum Austausch von Wissen, Erfahrungen sowie Lehr- und Lerninhalten. Das vereinfacht den Zugang zur Roberta-Community und unterstützt die Zusammenarbeit zwischen den Lehrkräften im Roberta-Netzwerk.

Das Roberta-Portal dient sowohl der Verbreitung der Kurse als auch der Unterstützung der KursleiterInnen. Es soll den regionalen wie auch den überregionalen Erfahrungsaustausch fördern. Die neuen Community-Funktionen des Portals machen dies möglich: Im Forum können die Lehrkräfte Erfahrungen austauschen und sich gegenseitig Tipps zum Durchführen der Kurse geben.

Bei akuten Fragen und Problemen finden sie schnell Antworten in den FAQs und im Roberta-Forum. Gefördert wurde die Entwicklung des Portals von der Fraunhofer Academy[2].

Die Roberta-Community ist ein zentraler Baustein des Portals. Sie stärkt die Gemeinschaft des Roberta-Netzwerks und macht alle Akteure sichtbar, so dass die richtige Ansprechperson immer schnell gefunden wird und auch direkt erreichbar ist. Das Roberta-Portal wird kontinuierlich von Fraunhofer IAIS weiterentwickelt.

2 Die Fraunhofer Academy ist eine Einrichtung der Fraunhofer-Gesellschaft, die in Kooperation mit ausgewählten und renommierten Partneruniversitäten und Partnerhochschulen Weiterbildung in ausgewählten Technologiebereichen anbietet. Fach- und Führungskräfte externer Unternehmen und öffentlichen Einrichtungen können auf diese Weise von der Forschungstätigkeit der Fraunhofer-Institute profitieren.

Abbildung 1.4.: Die aktuelle Startseite des Roberta-Portals

Ermöglicht wurde die Entwicklung des Roberta-Portals durch die Fraunhofer Academy. Ziel der Förderung durch die Fraunhofer Academy ist es, ein zukunftsfähiges Online-Portal aufzubauen, um die Roberta-Weiterbildungsmaßnahmen zu begleiten und zu unterstützen. Das Portal soll die Nutzung und Integration neuer Medien und Portaltechnologien zur effizienteren Betreuung des Roberta-Netzwerks unterstützen.

Dies bedeutet konkret:

- Online-Begleitung der Schulungsmaßnahmen – von der Schulungsanmeldung bis zur Zertifikatausstellung und Registrierung als Roberta-Teacher
- Verknüpfung der Portalinhalte mit neuen Medienarten – z.B. News-Feed als Applikation für Smartphones und Tablet-PCs
- Angebot und Bereitstellung von Lern-/Lehrvideos als integraler Bestandteil des Portals ebenso wie für Smartphones und Tablet-PCs
- Erweiterung und Integration einer 3D-Roboter-Lernanimation zur Kursvorbereitung von Roberta-Kursen in das Online-Portal

Weiterhin soll das Roberta-Portal als leistungsfähiges Webportal mit Community-Funktion etabliert werden:

- Forum/FAQs
- Mediasharing: Roberta-Lehrende und -Lernende können eigene Inhalte einstellen und bewerten.

- Internationalisierung des Portals: Über das Menü wird zu aktuellen Entwicklungen, zum Roberta-Konzept, zu Roberta-Angeboten und zum Roberta-Netzwerk informiert.

Hinweise zur Benutzung des Portals für Lehrkräfte

Was muss ich tun, wenn ich Roberta-Teacher werden will?

Nehmen Sie an einer Roberta-Basis-Schulung teil. Roberta-Basis-Schulungen gibt es sowohl für das LEGO® MINDSTORMS® EV3-System als auch für das LEGO® MINDSTORMS® NXT-System. Auf Nachfrage und bei entsprechend hoher Teilnehmerzahl können auch Roberta-Schulungen für das LEGO® MINDSTORMS® RCX-System durchgeführt werden. Angeboten werden die Basis-Schulungen an folgenden Standorten:

- Fraunhofer IAIS – Sankt Augustin/Bonn
- Zentrale der Fraunhofer-Gesellschaft – München
- Johannes-Kepler Gymnasium – Garbsen/Hannover
- Fachhochschule Kiel – Kiel
- Wolfgang-Borchert-Schule – Berlin.
- htw saar – Saarbrücken

Auf Nachfrage und bei einer Teilnehmerzahl von mindestens acht Personen werden auch Schulungen an anderen Orten von den Roberta-Coaches durchgeführt.

Sind Roberta-Schulungen kostenpflichtig?

Pro Teilnehmerin/Teilnehmer kostet die Schulung 290€. In den Schulungskosten sind die Roberta-Bände »Grundlagen« und »Experimentieren mit EV3« sowie ein USB-Stick enthalten.

Wie melde ich mich zu einem Kurs an?

Über das Roberta-Portal können Sie eine geeignete Schulung auswählen und sich direkt anmelden.

Wie bestelle ich Baukästen für meine Schule?

Nach erfolgreicher Teilnahme an einer Roberta-Basis-Schulung können Sie sich im Roberta-Portal registrieren. Mit der Anmeldung erhalten Sie Zugang zum Roberta-Bestellschein. Mit diesem Bestellschein können ausgewählte LEGO® MINDSTORMS® -Produkte bei unserem Kooperationspartner LEGO® Education bestellt werden.

Wann darf ich Roberta-Kurse durchführen?

Nach der Teilnahme an einer Roberta-Basis-Schulung.

Wo finde ich Downloads?

Roberta-Materialen können im internen Bereich des Portals heruntergeladen werden. Dieser Bereich ist ausschließlich zertifizierten und registrierten Roberta-Lehrkräften zugänglich. Einige, wenige Materialien sind auch öffentlich unter: www.roberta-home.de/angebot/materialien als kostenlose Downloads verfügbar.

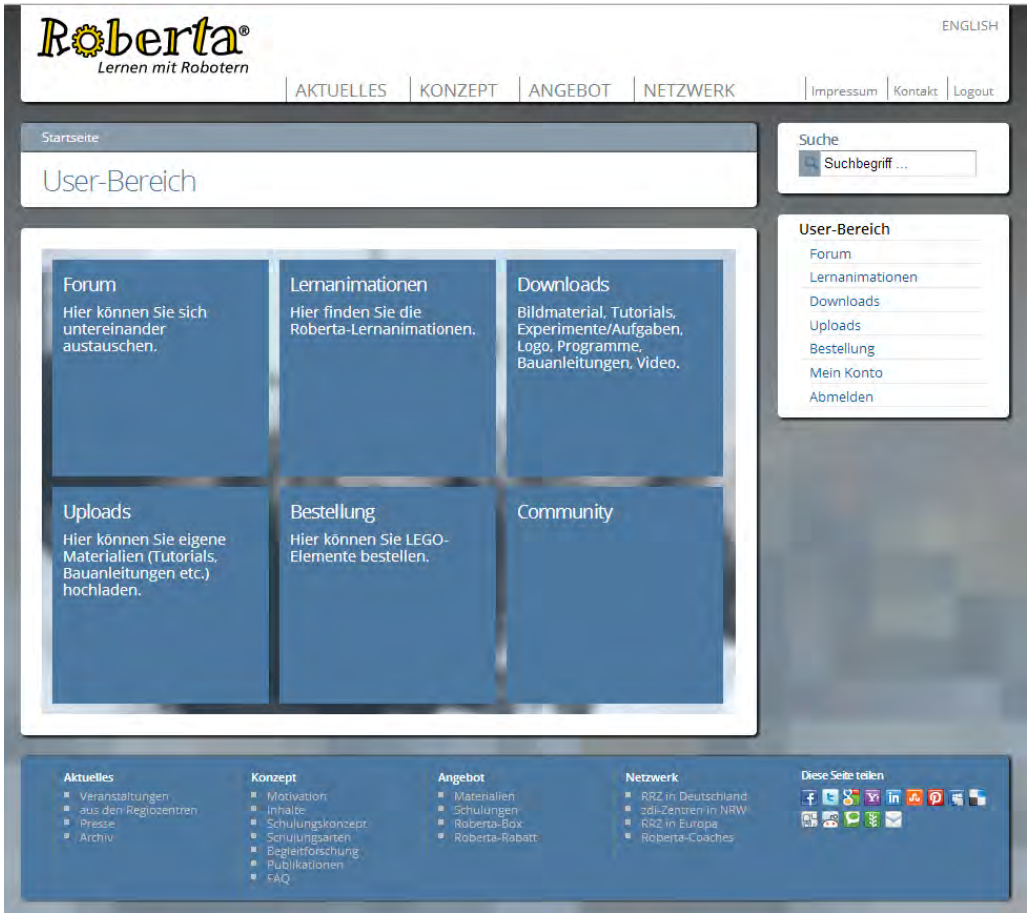


Abbildung 1.5.: Der interne Bereich des Roberta-Portals

Wie kann ich mich mit anderen Lehrkräften austauschen?

Im internen Bereich des Roberta-Portals gibt es ein Forum zum Austausch innerhalb des Roberta-Netzwerks.

Zertifizierte Roberta-Teacher können sich im Portal regis-

trieren. Damit erhalten Sie exklusiven Zugang zu weiteren, kostenfreien Roberta-Materialien sowie zum Roberta-Bestellschein, der Roberta-Lehrkräften einen Rabatt auf ausgewählte LEGO® MINDSTORMS® -Produkte bei LEGO® Education ermöglicht.

Roberta-Netzwerk

Das Roberta-Netzwerk in Deutschland ist geprägt von RobertaRegioZentren, die in ihrem unmittelbaren Einzugsbereich wirken und mit Schulen und Lehrkräften vor Ort zusammenarbeiten. Zurzeit (Stand: Juli 2014) gibt es 24 aktive RobertaRegioZentren in Deutschland, von denen die Zentren in Magdeburg und Koblenz-Landau seit Beginn der Entwicklung dabei sind. Weitere RobertaRegioZentren in den Bundesländern sind im Aufbau.

In **Nordrhein-Westfalen** gibt es die Initiative Zukunft durch Innovation.NRW, kurz: zdi. Diese ist eine Gemeinschaftsoffensive zur Förderung des naturwissenschaftlich-technischen Nachwuchses in Nordrhein-Westfalen. Von 2008 bis 2013 war Roberta Bestandteil von zdi. Hier fand eine Zusammenarbeit mit der vom Bildungsministerium des Landes NRW finanzierten zdi-Initiative statt, wobei Schulen und andere Bildungseinrichtungen sich zu zdi-RobertaZentren qualifizieren konnten.

Seit 2005 gibt es in **Berlin** den sogenannten eEducation Berlin Masterplan. Dies ist eine Initiative der Senatsverwaltung für Bildung, Jugend und Wissenschaft. Zielstellung des Masterplans sind die Ausbreitung der informationstechnischen Bildung in Berlin und die Erhöhung der Medienkompetenz. Roberta ist seit 2005 Teil des eEducation Berlin Masterplans. Im Rahmen dessen werden die Kosten für die an Roberta-Schulungen teilnehmenden Lehrkräfte von der Senatsverwaltung übernommen. Darüber hinaus erhalten die beteiligten Schulen eine Grundausstattung mit 3 Laptops und 3 LEGO® MINDSTORMS® Baukästen.

Im **Saarland** gibt es seit 2008 eine Initiative der Landesregierung zur Förderung des Einsatzes von Robotik an saarländischen Schulen. Sie wird durchgeführt vom Embedded Robotics Lab (EmRoLab) der Hochschule für Technik und Wirtschaft in Saarbrücken – (htw saar). Dort werden Lehrkräfte nach Roberta-Kriterien ausgebildet und die Schulen bekommen über die Unterstützung der Landesregierung Baukästen. Jährlich findet eine Roberta LEGO® Engineering Conference für interessierte Lehrkräfte statt. Unterstützt wird die Initiative zudem von ME-Saar (Verband der Metall- und Elektroindustrie des Saarlandes) sowie von regionalen Wirtschaftsunternehmen.

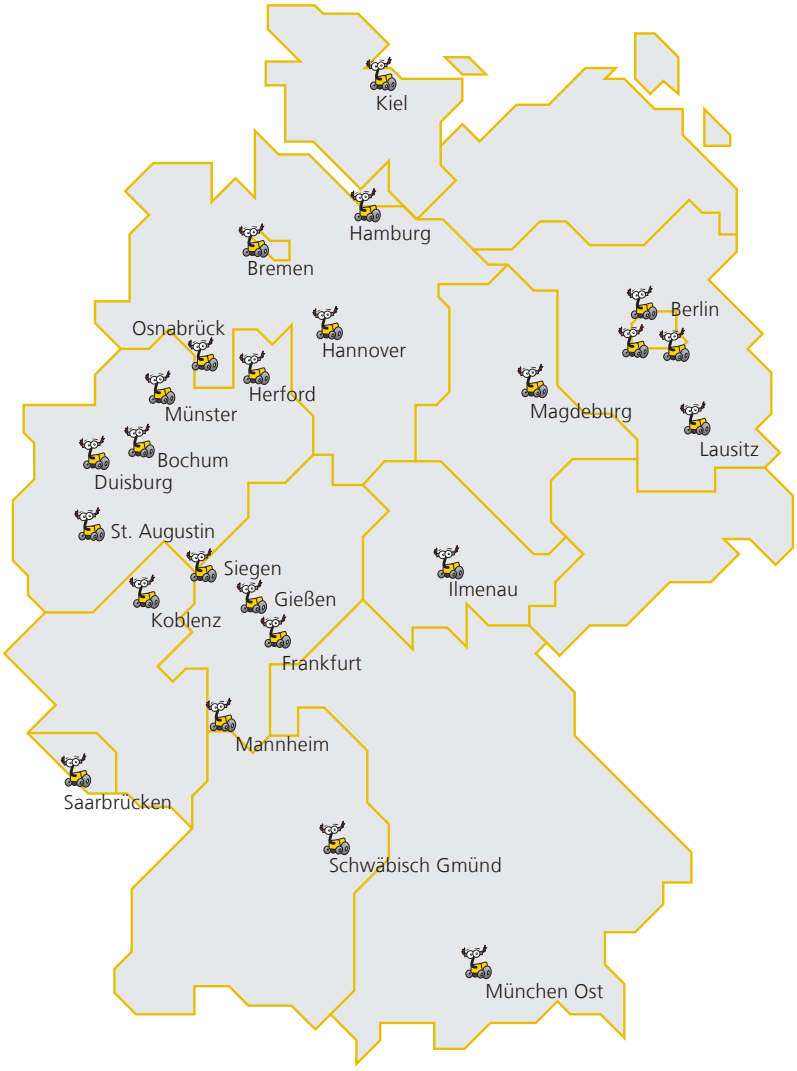


Abbildung 1.6.: RobertaRegioZentren in Deutschland

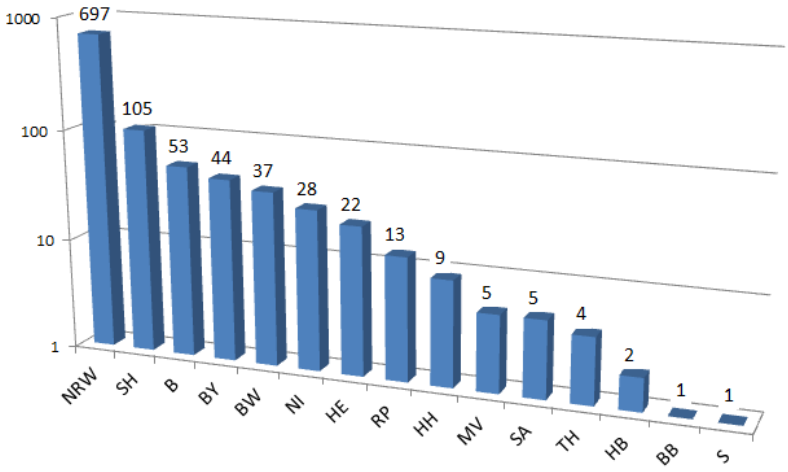


Abbildung 1.7.: Anzahl der Roberta-Teacher in den Bundesländern[3]

Im Rahmen des Projektes »Transfer Wissenschaft Schule« baut das Ministerium für Bildung und Wissenschaft des Landes Schleswig-Holstein auch in **Schleswig-Holstein** ein Roberta-Netzwerk auf. Dazu wurden 2013 in über 35 Pilotschulen aus dem Bereich der Regionalschulen, Gemeinschaftsschulen und Gymnasien jeweils die Ausbildung eines Roberta-Teachers sowie die Grundausstattung für einen Klassensatz Roboter mit 10 Roberta-Boxen finanziert.

Zum Netzwerk gehören auch die von Fraunhofer IAIS akkreditierten Roberta-Coaches, die Roberta-Schulungen zur Zertifizierung von Lehrkräften durchführen dürfen. Sie arbeiten eng mit der Zentrale zusammen und sind für die Durchführung der Roberta-Schulungen hauptverantwortlich.

Abbildung 1.7 zeigt, wie bedeutend eine Länderförderung für die Verbreitung von Roberta in den Bundesländern ist[4].

Das EU-Projekt »Roberta goes EU« hat dazu beigetragen, dass Roberta inzwischen auch in anderen europäischen Ländern mit pilotierten RobertaRegioZentren vertreten ist. Beteiligt waren fünf Länder, in denen sich die Verbreitung – in manchen mehr und in anderen weniger – fortgesetzt hat. Sehr aktiv ist Roberta in Österreich und der Schweiz.

3 Logarithmische Darstellung

4 Da im Saarland die Organisation der Roberta-Teacher in einem anderen strukturellen Kontext erfolgt, sind hier keine Zahlen für das Saarland enthalten.

Roberta-Schulen

Das Roberta-Konzept konzentriert sich nicht auf eine Schulform. Roberta ist schulübergreifend einsetzbar. Im Roberta-Netzwerk sind Roberta-Teacher von der Grundschule bis zur Hochschule tätig. Zurzeit (Stand: Juli 2014) sind Roberta-Teacher deutschlandweit an über 600 Schulen aktiv. Wie Abbildung 1.8 zeigt, bilden Gymnasien den größten Anteil, gefolgt von Gesamtschulen und Realschulen. Aber auch Hauptschulen sind mit 31 Schulen im Roberta-Netzwerk gut vertreten.

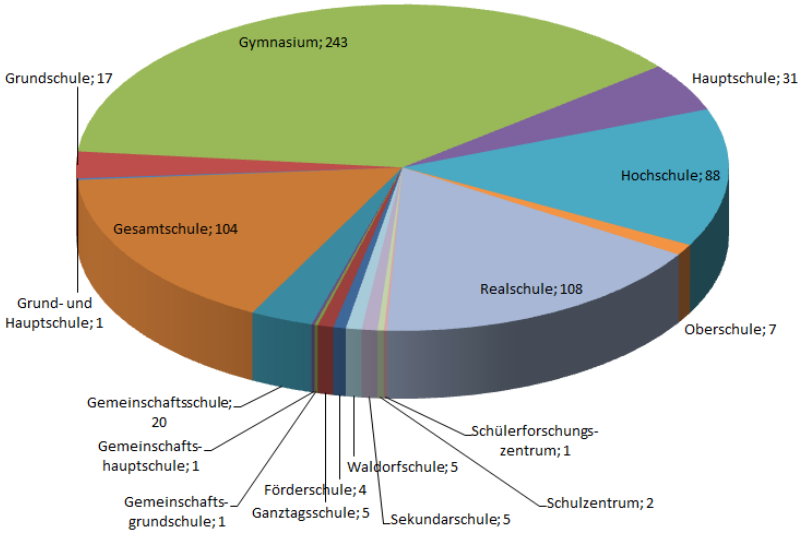


Abbildung 1.8.: Roberta-Teacher an verschiedenen Schulformen; Stand: Juli 2014

Roberta-Band: EV3-Programmieren mit Java

Das vorliegende Buch »EV3-Programmieren mit Java« wendet sich insbesondere an mit einer Programmiersprache erfahrene Multiplikatoren aus dem MINT-Sektor, die sich für die Programmierung des LEGO MINDSTROMS EV3-Roboterbaukasten mit Java interessieren. In vielen Gesprächen mit Lehrkräften während der vergangenen Jahre hat sich gezeigt, dass nach der Einführung der Programmierung mit Hilfe einer grafischen Programmiersprache viele Jugendliche den nächsten »Programmierschritt« machen möchten. Java als weit verbreitete objektorientierte und plattformunabhängige Programmiersprache, die eine hohe Verbreitung in Wirtschaft, Industrie, aber auch auf über 3 Milliarden Mobiltelefonen weltweit (Oracle 2014) hat, ist sehr gut für den Einsatz in Schulen geeignet. Durch die Nutzung des für LEGO MINDSTORMS entwickelten Java-Betriebssystems »leJOS« kann auch seit Anfang 2014

das EV3-System der LEGO MINDSTORMS-Serie mit Java programmiert werden.

Ergänzend zu diesem Band gibt es weitere, baukastenspezifische Roberta-Bände. Die nachfolgende Tabelle führt die Lehrmaterialien auf und zeigt, für welche Robotersysteme ein Roberta-Band derzeit verfügbar ist.

Roberta-Band	EV3	NXT	RCX
Grundlagen	x	x	x
Experimente	-[1]	x	x
Geschöpfe aus einem Baukasten	x	x	-
Programmiersprache Java	x	x	x
Programmiersprache NXC	-	x	x
Programmiersprache LabVIEW	-	x	-
Android AppInventor	-	x[2]	-
FIRST®LEGO®League	x	x	x

Tabelle 1.3.: Lehrmaterialien der Roberta-Reihe

Die gesamte Roberta-Reihe bietet eine Fülle an Anschauungsmaterialien: von einfachen Aufgaben, die maximal einen Sensortyp erfordern, über zusammengesetzte Aufgaben, die Kenntnisse aus den Übungen mit einfachen Aufgaben voraussetzen, bis hin zu Aufgaben, die die Beschäftigung mit einem Fachthema voraussetzen und strategische Überlegungen erfordern.

Nach der kurzen Einführung in das Roberta-Konzept wünschen wir Ihnen nun viel Freude beim Lesen unseres neuen Roberta-Bands: »EV3-Programmieren mit Java«.

Thorsten Leimbach, Beate Jost
Leitung der Roberta-Initiative

1 In Planung
2 Voraussichtlich erhältlich ab November 2014

Einführung

Für LEGO MINDSTORMS-Roboter gibt es verschiedene Programmiersprachen und Programmierumgebungen. Zum Beispiel grafische Sprachen wie NXT-G (für LEGO MINDSTORMS NXT), RIS (für LEGO MINDSTORMS RCX) und EV3-G (für LEGO MINDSTORMS EV3), die einen sehr leichten Einstieg in die Programmierung der LEGO MINDSTORMS-Systeme geben.

Seit November 2014 existiert auch die cloudbasierte, grafische Programmierumgebung »Open Roberta Lab«. Diese ermöglicht die Programmierung von LEGO MINDSTORMS EV3-Robotern ohne technische Hürden aus dem Browser heraus und ist unter www.open-roberta.org zu finden.

In diesem Roberta-Band: »EV3-Programmieren mit Java« wird die objektorientierte Programmiersprache Java für das LEGO MINDSTORMS EV3-System vorgestellt, denn Java bietet einen größeren Funktionsumfang als seine grafischen Verwandten, was die Programmierung von komplexeren und leistungsfähigeren EV3-Robotern ermöglicht.

Es wird einführend das Programmierparadigma der objektorientierten Programmierung (kurz OOP) an Hand einfacher Java-Beispiele behandelt. Um diese Beispiele am eigenen Rechner nachvollziehen zu können, muss zunächst eine funktioniere Entwicklungsumgebung vorhanden sein. Dazu sollte Anhang A durchgearbeitet werden. Dort wird die Installation der Entwicklungsumgebung Eclipse auf dem eigenen Rechner sowie die Installation von leJOS auf dem EV3-Stein Schritt für Schritt beschrieben.

Die in diesem Band vorgestellten JAVA-Beispielprogramme (und mehr) können im Roberta-Portal heruntergeladen werden: www.roberta-home.de/javaband-ev3.

Die Abkürzung leJOS steht für »LEGO Java Operating System« und ist ein Framework das u.a. die Firmware, die benötigt wird, um den EV3 mit Java programmieren zu können, enthält.

Java und leJOS sind kostenlos erhältlich und verfügen beide über eine große Gemeinschaft von EntwicklerInnen und AnwenderInnen, die sich für eine stetige Weiterentwicklung verantwortlich zeigen. Entwicklungsumgebungen wie Eclipse bieten für die Programmierung des EV3 alle üblichen Funktionalitäten wie Syntaxvervollständigung und -highlighting sowie Versionsverwaltung.

Die Einführung in die OOP, wie auch die aufgeführten Programmbeispiele, richten sich an interessierte LeserInnen, die bereits Erfahrungen mit der Programmierung von LEGO MINDSTORMS mit anderen Programmiersprachen gesammelt haben. Da dieser Band nicht das komplette Spektrum der objektorientierten Programmierung sowie deren Programmierparadigma abdecken kann, wird an dieser Stelle auf eine weiterführende, vertiefende Literaturliste am Ende dieses Buches verwiesen.

Was ist der EV3?

EV3 bezeichnet die dritte Generation des LEGO MINDSTORMS-Systems und ist ein von LEGO eingetragenes Warenzeichen. Dabei handelt es sich um einen programmierbaren 32-bit ARM9-Mikrocontroller vom Hersteller Texas Instruments mit einem auf Linux basierenden Betriebssystem. Dieser wird auch EV3-Brick oder EV3-Stein genannt. Die Taktrate beträgt 300 MHz und die Größe des Hauptspeichers beträgt 64 MB. Als Kommunikations- bzw. Programmierschnittstelle verfügt der EV3 über einen USB-2.0-Anschluss und ein Funkmodul das Bluetooth als Standard verwendet. Der Baustein verfügt über vier Motoranschlüsse (Port A bis D) und vier Sensoranschlüsse (Port 1 bis 4).

Abbildung 2.1.: EV3-Stein

Was ist leJOS?

Bei leJOS handelt es sich um eine schlanke Java Virtual Machine (JVM), die seit 2013 für den EV3-Stein verfügbar ist. leJOS begann ursprünglich als Hobby-Open-Source Projekt von José Solórzano im Jahr 1999 unter dem Namen TinyVM. Der Name ist eine Kombination aus dem Akronym »JOS« (für »Java Operating System«) und dem spanischen Wort »lejos« (auf deutsch: »weit entfernt« oder »weit«). Durch leJOS ist es möglich den EV3 mit Hilfe der Hochsprache Java zu programmieren. Ebenso trägt auch die Klassenbibliothek, mit der die Komponenten des EV3 (Motoren, Sensoren etc.) angesprochen werden, den Namen leJOS. Damit stellt die leJOS-Software ein Java-Betriebssystem als Alternative zur vorinstallierten LEGO-Firmware für den EV3 dar. leJOS wird von einer bootbaren microSD-Karte gestartet, ohne dabei die auf dem EV3 vorhandene LEGO-Software zu verändern bzw. zu löschen (Siehe Anhang A).

Vorteile von leJOS

Durch die Erweiterung mit leJOS können die Vorteile der Objektorientierung für den EV3 genutzt werden. Es ist mit ein wenig Aufwand in professionelle Entwicklungsumgebungen integrierbar und arbeitet schneller und wesentlich ressourcenschonender als die grafische Lösung EV3-G. Außerdem ist unter leJOS die Fließkommaarithmetik implementiert. Zusätzlich bietet leJOS eine vollständige Bluetooth-Unterstützung und ist leicht erweiterbar, beispielsweise um Klassen für Sensoren von Drittherstellern zu nutzen. Des Weiteren unterstützt leJOS High-Level-Robotik-Tasks (Navigation, Localization etc.) und besitzt eine große Community (inklusive Wiki und gut dokumentierter API).

leJOS stellt zudem ein vollständiges Java-Runtime-System zur Verfügung. Dies beinhaltet die folgenden Features:

- Objektorientierte Programmierung (Java)
- Multithreading
- Arrays (auch multi-dimensionale Arrays)
- Rekursion
- Synchronisation
- Exceptions
- Java Datentypen (inklusive `float`, `long` und `String`)
- Die meisten der Java-Klassen `java.lang`, `java.util` und `java.io`

Anmerkung zu leJOS

Da es sich bei leJOS in der von uns verwendeten Version 0.9.0 noch um eine Beta-Version handelt und diese ständig weiterentwickelt wird, empfiehlt es sich unter http://lejos.sourceforge.net die Tutorials von Zeit zu Zeit durchzusehen. Die im Folgenden beschriebenen Komponenten wurden erfolgreich getestet.

IDE

Bei einer IDE (Integrated Development Environment) handelt es sich um ein Programm, das die EntwicklerInnen möglichst umfassend beim Erstellen von Quelltexten unterstützt. Dazu besteht die IDE üblicherweise aus einem Editor, einem Compiler, Werkzeugen zur Programmerstellung und einem Debugger. Der Editor bietet in der Regel Syntaxhighlighting und -vervollständigung. Somit wird der gesamte Entwicklungsprozess effizienter. Der *Compiler* wandelt das (Quell-) Programm in ein für das benutzende System äquivalentes (Ziel-) Programm um. In den meisten Fällen handelt es sich hierbei um die Übersetzung eines in Assemblersprache, Bytecode oder Maschinensprache geschriebenen Programms. Der *Debugger* ist ein Hilfsmittel, das bei der Auffindung, Diagnose und Behebung von Fehlern im Programmcode von der IDE bereit gestellt wird.

Eclipse IDE

Eclipse ist eine IDE, die nachfolgend kurz beschrieben und für die Programmierung des EV3-Steins mit Java verwendet wird.

Die Entwicklung von Eclipse begann 2001 initiiert von IBM. Ursprünglich wurde Eclipse nur als IDE für Java genutzt. Durch mittlerweile viele Erweiterungen, sowohl kommerziell als auch nicht-kommerziell, werden aber auch viele andere Programmiersprachen wie C/C++, Python und einige weitere unterstützt. Im Jahr 2004 wurde auf Beschluss von IBM die eigenständige »Eclipse Foundation« gegründet, die seither für die Entwicklung von Eclipse zuständig ist. Seit der Version 3.0 wird Eclipse nicht mehr als erweiterbare IDE angeboten, sondern stellt den Kern der Entwicklungsumgebung bereit, der seine vielzähligen Funktionen durch die z.T. automatisch geladenen Plug-ins erhält. Auch für leJOS ist es ein solches Plug-in erhältlich.

Auf der Eclipse-Webseite werden die aktuellen Versionen in verschiedenen Paketen angeboten, in denen bereits auf die jeweilige Aufgabe zugeschnittene Plug-ins enthalten sind. Standardmäßig wird Eclipse in Englisch dargestellt, es ist jedoch auch möglich, durch Sprach-Plug-ins eine deutsche Benutzeroberfläche zu erhalten.

Vergleich Programmiersprachen

Um den EV3-Stein zu programmieren gibt es verschiedene Programmierumgebungen und Sprachen. Die von LEGO mitgelieferte Software wird schlicht als »EV3-Software« bezeichnet. Sie ist an die NXT-Software »NXT-G« angelehnt und richtet sich an EinsteigerInnen. Sie bietet eine grafische Oberfläche und fertige Blöcke für Programmabläufe. So kann z.B. die Motoransteuerung mit einem Mausklick implementiert werden. Mit der Programmiersprache NEPO – die im »Open Roberta Lab« verwendet wird – steht ebenfalls eine grafische Programmiersprache zur Verfügung, mit der ein unkomplizierter Einstieg in die Programmierung von EV3-Robotern möglich ist. Eine weitere Programmiersprache ist NXC (Not eXactly C). Die Syntax orientiert sich an der Programmiersprache C und kann z.B. mit der Programmierumgebung BricxCC eingesetzt werden. Die Sprache RobotC, entwickelt von der Carnegie Mellon University in Pittsburgh, Pennsylvania, hingegen orientiert sich nicht nur wie NXC an C, sondern übernimmt die Syntax vollständig. RobotC ist eine kostenpflichtige Programmierumgebung, in der RobotC programmiert werden kann. Der Vorteil dieser Sprache ist der große Funktionsumfang und die ausgereifte Programmierumgebung, die eine einfache Debug Möglichkeit und ein Feature bietet, mit dem die Sensoren kontinuierlich abgefragt werden können.

In Tabelle 2.1 werden die Vor- und Nachteile sowie die Merkmale der oben genannten Programmiersprachen dargestellt.

	Java mit leJOS	EV3-Software	RobotC	NEPO
Installation	+	++	+	+++
Handhabung	+	++	+	++
Kosten	kostenlos	kostenlos	49$[1]	kostenlos
Einstieg	0	++	+	+++
Funktionsumfang	++	+	++	++

0 = neutral; + = gut; ++ = sehr gut; +++ = hervorragend

Tabelle 2.1.: Vor- und Nachteile der Programmiermethoden

1 für 365-Tage-Lizenz

Plattformunabhängigkeit

Die Verwendung von Java als Programmiersprache ermöglicht weitestgehende Plattformunabhängigkeit. Das bedeutet Java kann sowohl unter Windows, Linux als auch Mac OS oder auf anderen technischen Systemen genutzt werden, vorrausgesetzt eine Java Virtual Machine (JVM) existiert dafür.

Der Quellcode kann mit jedem beliebigen Editor oder einer IDE (Integrated Development Enviroment geschrieben werden. Die erstellte Datei wird mit dem Java-Compiler (javac) in Bytecode übersetzt (siehe Abbildung 2.2). Diese Dateien haben die Erweiterungen .class und werden von der virtuellen Maschine (VM) interpretiert. Diese VM gibt es für alle gängigen Rechner- oder Hardwareplattformen sowie für mobile Endgeräte wie zum Beispiel Mobiltelefone. Mit leJOS existiert eine solche virtuelle Maschine auch für den EV3.

Die VM wird (zumeist vom Hersteller) auf dem Gerät implementiert und speziell daran angepasst. Die Java-EntwicklerInnen hingegen programmieren auf dieser standardisierten Plattform, die auf allen Geräten identisch ist, unabhängig von den technischen Details des Geräts selbst. Die folgende Grafik zeigt den generellen Aufbau eines Java-Programms mit der virtuellen Maschine.

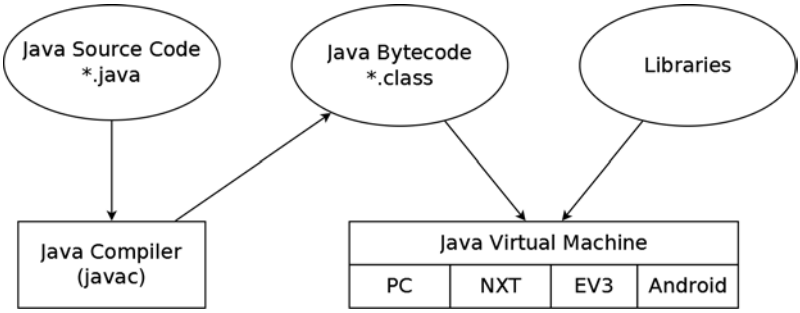


Abbildung 2.2.: Funktionsweise der Java Virtual Machine

Kapitel 3

Objektorientierte Programmierung

In diesem Kapitel werden die Grundlagen der Objektorientierten Programmierung beschrieben und ein Einstieg in Java geliefert. Fortgeschrittene LeserInnen, die schon Erfahrung in der Java-Programmierung haben und durch die Lektüre dieses Buches lediglich den Umgang mit leJOS und dem EV3 erlernen wollen, können wie folgt vorgehen:

- Durcharbeiten von Anhang A: Installation
- Lesen von Kapitel 6 ab Seite 73: Erste Schritte in leJOS
- Durchgehen von Anhang B

Sowohl der EV3 als auch der verwendete Computer sind an dieser Stelle vollständig eingerichtet und es können die EV3-Beispielprogramme aus Kapitel 4, Kapitel 5, Kapitel 7 sowie aus Kapitel 8 übertragen und ausgeführt werden. Das Nachvollziehen dieser Beispielprogramme und die Beschreibung der verfügbaren leJOS-Klassen in Kapitel 7 liefern eine gute Grundlage um mit der Programmierung eigener Roboter zu beginnen.

Das im Internet kostenlos einsehbare Buch Galileo Openbook »Java ist auch eine Insel« kann als allgemeine Java-Grundlektüre empfohlen werden. Auch wenn sich »Java ist auch eine Insel« nicht konkret mit leJOS und dem EV3 beschäftigt, werden hier wichtige Grundlagen für die Programmierung mit Java anschaulich und leicht verständlich beschrieben. (Ullenboom, Christian 2011)

3.1 Allgemeines zur Objektorientierung

Mit dem Begriff »Objektorientierung« wird eine Art der Modellbildung in der Softwareentwicklung beschrieben. Unter dieser ist, sehr vereinfacht ausgedrückt, die Abstraktion der realen Welt und der Transfer dieses abstrahierten Modells auf eine Maschine zu verstehen. Es ist dennoch weiterhin möglich, prozedural (Aufteilung von Aufgaben und Teilproblemen in Prozeduren) zu programmieren, wie es zum Beispiel in C gemacht wird. Das Konzept der Objektorientierung (kurz: OO) bietet viele Vorteile, die die Entwicklung von Software von Grund

auf verändert und erleichtert. Dabei wird versucht auf menschliche Strukturierungs- und Klassifizierungsmethoden zurückzugreifen.

Der Objektorientierung liegt eine Aufteilung der zu beschreibenden Welt in Objekte mit ihren Eigenschaften und Operationen zugrunde. Realisiert wird diese Aufteilung durch das Konzept der Klasse, bei dem Objekte aufgrund ähnlicher Eigenschaften zusammengefasst und nach außen gekapselt werden. Die Struktur eines Objekts wird durch die Attribute (auch Eigenschaften) seiner Klassendefinition festgelegt. Das Verhalten des Objekts wird von den Methoden der Klasse bestimmt.

Vorteile der Objektorientierung

Ein Vorteil des auf der Objektorientierung beruhenden Programmierparadigmas, der Objektorientierten Programmierung (kurz: OOP), stellt die Möglichkeit dar, Programmteile wiederzuverwenden. Konsistent entwickelte Programmteile können in verschiedenen Konfigurationen und verschiedenen logischen Zusammenhängen genutzt werden. Ein weiterer Vorteil ist, dass das zu erzeugende Programm in übersichtliche Programmstücke geteilt werden kann. Deren Erstellung und Wartung lassen sich dadurch besser und transparenter strukturieren.

Die unterschiedlichen Programmteile sind somit unabhängig voneinander und können einzeln und von verschiedenen Personen entwickelt werden. Um von den beschriebenen Vorteilen zu profitieren, werden die vier folgenden Konzepte genutzt:

- Generalisierung
- Vererbung
- Kapselung
- Polymorphie

Diese vier Konzepte werden in Kapitel 4 ab Seite 53 beschrieben.

3.2 Grundsätzliche Strukturen der OOP

Bevor auf diese Konzepte eingegangen werden kann, werden die grundsätzlichen Strukturen der objektorientierten Programmierung vorgestellt.

Klassen

Wie eingangs erwähnt, lehnt sich das Konzept der OOP stark an Strukturierungs- und Klassifizierungsmethoden aus der alltäglichen (menschlichen) Betrachtungsweise unserer realen Welt an. Das folgenden Beispiel soll dies verdeutlichen:

Der Begriff PKW stellt eine Klassifizierung von motorisierten Vehikeln dar, die (meist vierrädrig) zur Personenbeförderung von A nach B dienen. Der Begriff legt dabei aber weder die Anzahl der Türen, noch Farbe, Form, Hubraum etc. des PKWs fest.

Ein ähnliches, auf die Thematik dieses Buches bezogenes Beispiel bietet der Begriff Roboter:

Als Roboter wird im herkömmlichen Sinn eine technische Apparatur bezeichnet, die dazu konzipiert wurde, Aufgaben des Menschen zu übernehmen, um diesen zu entlasten. Dabei können viele verschiedene Arten beispielsweise im Aussehen unterschieden werden: humanoider Roboter, Industrieroboter, mobiler Roboter etc. Auch weitere Eigenschaften wie z.B. Einsatzgebiet, Kosten, Aufgabe oder die Achsenanzahl sind nicht definiert.

Eine interessante Übersicht mit Links zu Robotikanwendungen sortiert nach Einsatzgebieten liefert: https://robots.zeef.com/roberta.roboter0

Diese Art der Strukturierung greift die objektorientierte Programmierung durch die Verwendung von Klassen und Objekten auf. Auf die Beispiele angewandt, werden die Klassen `PKW` und `Roboter` angelegt, um festzulegen welche Attribute ein später erzeugtes Objekt der Klasse besitzen soll. Das Objekt stellt dann mit konkreten Attributwerten etwas tatsächlich Existierendes dar, z.B. den gelben VW Käfer von Herrn Müller oder den Industrieroboter der Automobilfirma Ford.

Folgendes Beispiel zeigt, wie in Java die Klasse `ExampleRobot` definiert wird.

Programm 3.1: ExampleRobot-Klasse

```
1  public class ExampleRobot {
2      //variables:
3      public String name;
4      public double price;
5      public int yearOfPurchase;
6      public String appearance;
7
8      //methods:
9      public int age(int currentYear) {
10         return currentYear - yearOfPurchase;
11     }
12 }
```

Durch das Schlüsselwort `class` wird die Klassendefinition begonnen. In den geschweiften Klammern können beliebig viele Variablen und Methodendefinitionen erfolgen. Die Variablen (auch Attribute einer Klasse genannt) speichern die Eigenschaften der Klasse – hier Bezeichnung (`name`), Kosten (`price`), Kaufjahr (`yearOfPurchase`) und Aussehen (`appearance`). Die Methoden definieren die Fähigkeiten bzw. das Verhalten von Objekten. Im Beispiel kann über die Methode `age(currentYear)` das Alter des entsprechenden Roboters zurückgegeben werden. Methoden werden innerhalb von Klassen angelegt und können auf die Klassen-Attribute zugreifen. Der sogenannte Modifier `public` und weitere werden unter dem Punkt Kapselung in Kapitel 4 ab Seite 53 erklärt.

Da Software häufig von internationalen Teams entwickelt wird, bezeichnet man Variablen sowie Methoden und verfasst Quelltextkommentare auf Englisch. Wir schließen uns dieser Praxis an, im Text werden aber dennoch die deutschen Übersetzungen mit angegeben.

Objekte

Ein Objekt (oder auch die Instanz[1]) ist eine konkrete Ausprägung einer Klasse. Wird ein Objekt von einer Klasse erzeugt, so erhält es alle Attribute und Methoden, die in dieser Klasse enthalten sind. Des Weiteren haben alle Objekte drei wichtige Eigenschaften:

- Jedes Objekt hat einen Zustand.
- Jedes Objekt zeigt ein Verhalten.
- Jedes Objekt hat eine Identität.

Die aktuellen Werte der Variablen eines Objekts bilden seinen Zustand. Dieser kann sich im Verlauf eines Programms durchaus verändern. Das Verhalten eines Objekts, wird durch seine Methoden gebildet. Sie definieren unter anderem, wie das Objekt seinen oder den Zustand anderer Objekte verändern kann. Darüber hinaus besitzt jedes Objekt intern eine eigene, unveränderliche Identität. Auch wenn zwei Objekte einer Klasse den gleichen Zustand haben und gleiches Verhalten zeigen, sind es dennoch unterschiedliche Objekte mit unterschiedlichen Identitäten.

1 Instanz und Objekt werden in der Literatur oftmals synonym verwendet, wir schließen uns diesem Gebrauch innerhalb dieses Bandes ebenfalls an.

Erzeugen eines Objekts

Für das Erzeugen eines Objekts muss eine Variable vom Typ der Klasse (hier `ExampleRobot`) deklariert werden. Mit dem =-Operator wird dieser Variable dann ein neu erzeugtes Objekt zugewiesen. Die Erzeugung des Objekts erfolgt mit Hilfe des `new`-Operators, gefolgt vom Klassennamen mit zwei runden Klammern (also `ExampleRobot()`). Dieser Ausdruck wird Default-Konstruktor genannt und in Abschnitt 3.3 auf der nächsten Seite beschrieben.

Programm 3.2: Erzeugen des Objekts "myEV3"

```
1  public class ExampleRobot {
2      public String name;
3      public double price;
4      public int yearOfPurchase;
5      public String appearance;
6
7      public int age(int currentYear) {
8          return currentYear - yearOfPurchase;
9      }
10
11     public static void main(String[] args) {
12         ExampleRobot myEV3;
13         myEV3 = new ExampleRobot();
14     }
15 }
```

Die Erzeugung des Objekts erfolgt hier in der sogenannten `main`-Methode. Diese stellt das Hauptprogramm in einer Klassendatei dar und wird ebenfalls in Abschnitt 3.3 beschrieben.

Die Deklaration einer Variable und das Zuweisen eines Objekts kann auch kombiniert in einer Zeile erfolgen:
`ExampleRobot myEV3 = new ExampleRobot();`

Im letzten Beispielprogramm, gibt es ein Objekt der Klasse `ExampleRobot`. Dieser konkrete Roboter (`myEV3`) hat die allgemeinen Attribute der Klasse: Bezeichnung (`name`), Kosten (`price`), Kaufjahr (`yearOfPurchase`) und Aussehen (`appearance`). Aber er besitzt noch keine Ausprägungen, also einen undefinierten Zustand. Im Beispielprogramm 3.3 wird nun folgender Zustand hergestellt:

- Bezeichnung: LEGO EV3
- Kosten: 319,95
- Kaufjahr: 2013
- Aussehen: Roberta

Dies geschieht durch Verwendung des `.`-Operators, mit dessen Hilfe auf Variablen und Methoden eines Objekts zugegriffen werden kann.

Programm 3.3: Initialisieren der Objektvariablen

```
1  public class ExampleRobot {
2      public String name;
3      public double price;
4      public int yearOfPurchase;
5      public String appearance;
6
7      public int age(int currentYear) {
8          return currentYear - yearOfPurchase;
9      }
10
11     public static void main(String[] args) {
12         ExampleRobot myEV3;
13         myEV3 = new ExampleRobot();
14
15         myEV3.name = "LEGO EV3";
16         myEV3.price = 319.95;
17         myEV3.yearOfPurchase = 2013;
18         myEV3.appearance = "Roberta";
19     }
20 }
```

3.3 Generelle Programmstrukturen

Jedes Programm benötigt eine grundlegende Struktur. Genau wie in anderen Programmiersprachen ist diese Struktur fest vorgegeben und muss unbedingt beachtet werden. Dieses Kapitel soll einen kurzen Überblick über die verschiedenen Elemente geben, die in einer Klasse vorkommen müssen.

Dateistruktur

Die Programmstruktur eines leJOS-Programms gestaltet sich recht einfach. Es sollte darauf geachtet werden, einige Vorgaben und Konventionen einzuhalten. Es ist üblich in jeder Datei eine öffentliche Klasse (die mit Schlüsselwort `public` deklariert wird) zu erstellen. In der Datei können sich dann beliebig viele weitere Helferklassen befinden, die aber jeweils nicht öffentlich sind. Der Dateiname entspricht dem Klassennamen der öffentlichen Klasse. Die Klasse `ExampleRobot` wird also auch in der Datei `ExampleRobot.java` abgespeichert.

Paketimport

Jedes Java-Programm nutzt bestimmte Bibliotheken. Darin sind die Befehle beschrieben, die bei der Programmierung eingesetzt werden sollen. Bei der Programmierung mit leJOS müssen die Klassen für Sensoren und Motoren etc. aus den jeweiligen Packages eingebunden werden, damit der Programmcode von Java verstanden und richtig umgesetzt werden kann. Möchte man die Klasse KLASSE aus dem Package PACKAGE einbinden, so funktioniert dies nach folgendem Grundprinzip:

```
import PACKAGE.KLASSE;
```

Soll beispielsweise die Klasse »EV3ColorSensor« aus dem Package »lejos.hardware.sensor« zur Ansteuerung des EV3-Farbsensors verwendet werden, so lautet der Befehl dafür:

```
import lejos.hardware.sensor.EV3ColorSensor;
```

Möchte man direkt mehrere Klassen aus einem Package einbinden, zum Beispiel die leJOS-Klassen zur Ansteuerung aller Sensoren, wird der `*`-Operator verwendet:

```
import lejos.hardware.sensor.*;
```

Eclipse unterstützt die EntwicklerInnen beim Einbinden der Klassen durch einen Hinweis, falls Klassen verwendet werden sollen, die noch nicht eingebunden sind. Durch einen einfachen Linksklick auf den entsprechenden Hinweis fügt Eclipse die benötige `import`-Zeile automatisch zum Programmcode hinzu.

Klassendefinition

Nach den Paketimporten wird die eigentliche Klasse definiert. Mit den Schlüsselwörtern `public class <Klassenname>` wird eine neue öffentliche Klasse angelegt. In unserem Beispiel ist dies:

```
public class ExampleRobot
```

Danach beginnt der Klassenrumpf, der von geschweiften Klammern umschlossen wird (siehe zum Beispiel Programm 3.3).

Die main-Methode

Ist das Grundgerüst geschaffen, bedarf es nun noch der Hauptfunktion die den Namen `main` trägt. Auch diese Methode ist vom Typ `public` und da sie eine Klassenmethode ist, bekommt sie noch das Schlüsselwort `static`. In unserem Beispiel ist dies:

```
public static void main(String[] args)
```

Der `main`-Funktion muss immer ein Stringarray (also eine Feldvariable vom Typ String) übergeben werden. Übergeben wird das Feld `args[]`, in dem auf dem ersten Platz `args[0]` der Klassenname selber steht. Auf den folgenden Plätzen finden sich die Startparameter. In der Hauptfunktion `main` kann nun das gesamte Programm untergebracht, andere Objekte erstellt oder Threads (siehe Abschnitt 5.3 auf Seite 68) gestartet werden. Ist die `main`-Funktion abgearbeitet, endet automatisch auch die Programmausführung auf dem EV3. Ist eine Klasse jedoch nur dazu da, Objekte von ihrem Typ zu instanziieren, verfügt sie nicht über eine `main`-Methode. Stattdessen existiert dort ein sogenannter Konstruktor.

Konstruktor

Der *Konstruktor* wird bei jeder Instanziierung eines Objekts aufgerufen. Er kann als eine Art spezielle Methode ohne Rückgabewert bezeichnet werden. Diese Methode ist fest mit der Klasse verbunden und trägt ihren Namen. Mit der Ausführung des Konstruktors können beispielsweise Initialwerte gesetzt oder Zählvariablen erhöht werden. Aufgerufen wird der Konstruktor, wenn ein Objekt mit dem Schlüsselwort `new` erzeugt wird (siehe Programm 3.4). Damit wird der Konstruktor für jedes Objekt genau einmal aufgerufen. Ein nachträglicher oder erneuter Aufruf ist nicht möglich. Neben Konstruktoren gibt es in Java auch Destruktoren, die vor dem Zerstören eines Objekts aufgerufen werden. Da Java über eine automatische Speicherverwaltung verfügt, wird den Destruktoren in diesem Buch keine weitere Bedeutung beigemessen.

Im nächsten Beispielprogramm werden die Variablen des Roboters `myEV3` nicht in der `main`-Methode, sondern in einem eigens dafür erstellten Konstruktor initialisiert.

Programm 3.4: ExampleRobot mit Konstruktor

```
1  /**
2  * Example Robot-class.
3  * Object is being created in main-method
4  * and variables are initialized with a specific
       constructor.
5  */
6  public class ExampleRobot {
7      public String name;
8      public double price;
9      public int yearOfPurchase;
10     public String appearance;
11
12     public int age(int currentYear) {
13         return currentYear - yearOfPurchase;
14     }
15
16     ExampleRobot(String name, double price, int
           yearOfPurchase, String appearance) {
17         this.name = name;
18         this.price = price;
19         this.yearOfPurchase = yearOfPurchase;
20         this.appearance = appearance;
21     }
22
23     public static void main(String[] args) {
24         ExampleRobot myEV3;
25         myEV3 = new ExampleRobot("LEGO EV3", 319.95,
               2013, "Roberta");
26     }
27 }
```

Beim Erstellen des Objekts mit dem `new`-Operator werden dem Konstruktor die Variablenwerte als Parameter, getrennt durch Kommata, übergeben. Bei der Konstruktordefinition kann dann angegeben werden, was mit den so übergebenen Werten geschehen soll. In unserem Fall werden Sie den Variablen des Objekts zugewiesen. Damit es dabei nicht zu Verwechslungen zwischen Parameternamen und Variablennamen kommt, werden letztere durch den sogenannten `this`-Pointer gekennzeichnet. Dieser verweist immer auf das Objekt, zu dem die Methode (oder der Konstruktor) gehört.

Default-Konstruktor

Wird beim Anlegen einer Klasse kein expliziter Konstruktor angegeben, erzeugt der Compiler automatisch einen Default-Konstruktor, der ohne die Angabe von Parametern aufgerufen wird.

Java Codekonventionen

Die Java Codekonventionen stellen Regeln in Bezug auf die Form von Java Quelltext dar. Es ist ratsam sich an diese Regeln zu halten, da man sich so als Entwickler schneller in fremden Code einfinden kann und die Lesbarkeit erhöht wird. Die von Java herausgegebene Dokumentation zu den Codekonventionen kann unter http://www.oracle.com/technetwork/java/codeconvtoc-136057.html angesehen werden. Im folgenden werden die wichtigsten Konventionen vorgestellt:

- Der Code wird für jeden neuen Block um 4 Leerzeichen eingerückt. Eclipse rückt meist automatisch mit dem richtigen Abstand ein, ansonsten kann auch der Tabulator verwendet werden.
- Passt ein Ausdruck nicht auf eine Zeile, wird er nach einem Komma oder vor einem Operator umgebrochen.
- Pro Variablendeklaration sollte eine Zeile verwendet werden.
- Lokale Variablen sollten bei der Deklaration initialisiert werden (nur möglich, falls der Wert nicht noch berechnet werden muss).
- Die Initialisierung wird am Anfang eines Blocks platziert.
- Jede Zeile sollte höchstens eine Anweisung enthalten.
- Klassennamen sollten Substantive mit großem Anfangsbuchstaben sein. In Java wird generell der sog. CamelCase verwendet. Das bedeutet, dass interne Worte in einem Bezeichner wieder groß geschrieben werden. Zum Beispiel: KlassenNameImCamelCase.
- Schnittstellenbezeichner werden wie Klassennamen gewählt.
- Methoden sollten Verben sein und mit kleinem Anfangsbuchstaben beginnen. Auch bei Methoden wird der camelCase verwendet, allerdings mit kleinem Anfangsbuchstaben. Zum Beispiel: methodenNameImCamelCase.
- Variablennamen werden wie Methodennamen formatiert. Sie sollten kurz sein und sinnhaft auf die Funktion der Variable hinweisen.
- Konstantennamen bestehen ausschließlich aus Großbuchstaben. Einzelne Worte werden durch einen Unterstrich getrennt. Zum Beispiel: EINE_KONSTANTE.

Kommentare und Javadoc

In Java gibt es, wie in den meisten Programmiersprachen, mehrere Möglichkeiten seinen Quellcode mit Kommentaren zu versehen. Die Einfachste ist der sogenannte Zeilenkommentar. Dabei wird an beliebiger Stelle im Quelltext durch Einfügen von `//` ein Kommentar eingeleitet. Alle Zeichen die danach in dieser Zeile stehen werden nicht vom Compiler übersetzt. Der Kommentar endet mit der Zeile.

Die zweite Möglichkeit ist die Verwendung von sogenannten Blockkommentaren. Diese beginnen mit `/*` an beliebiger Stelle im Quelltext und enden mit `*/`. Alle Zeichen dazwischen werden vom Compiler ignoriert. Blockkommentare können sich auch über mehrere Zeilen erstrecken.

Kommentare können auch dazu verwendet werden um Programmteile zu Testzwecken auszukommentieren und damit nicht mit zu compilieren. Im Gegensatz zum Löschen der Programmzeilen können sie dann einfach wieder einkommentiert werden, falls sie wieder verwendet werden sollen.

Das Kommentieren von Quelltexten ist ein oft vernachlässigter, aber wichtiger Bestandteil der Softwareentwicklung, da Code in der Regel häufiger gelesen als geschrieben wird. Vor allem wenn Programmkomponenten anderen Entwicklern zur Verfügung gestellt werden sollen, müssen diese gut dokumentiert sein. Das Erstellen einer externen Dokumentation zum Quelltext wurde früher manuell gemacht. Dies war sehr aufwendig und fehleranfällig, da alle Änderungen im Quellcode auch in der Dokumentation umgesetzt werden mussten.

Heutzutage werden solche externen Dokumentationen automatisch, meistens im HTML-Format, erstellt. Bei Java übernimmt diese Aufgabe das Programm »Javadoc«. Dazu werden vor jeder Klasse, jedem Interface, jedem Konstruktor, jeder Methode und jedem Attribut spezielle Dokumentationskommentare verwendet. Diese beschreiben das entsprechende Element und werden von Javadoc in die Dokumentation umgewandelt. Dokumentationskommentare sind eine Sonderform der Blockkommentare. Sie beginnen mit `/**` und enden mit `*/`. Zeilen dazwischen beginnen meist mit einem `*`, was den Kommentar optisch aufwertet aber keine weitere Funktion erfüllt.

Der erste Satz jedes Dokumentationskommentars wird in der Zusammenfassung der Methoden und Attribute angezeigt. Dieser erste Satz sollte nach Möglichkeit die Funktionalität schon komplett beschreiben. Der gesamte Dokumentationskommentar wird dann in der Detailan-

sicht aufgeführt. Es sollte bei der Beschreibung der Methoden eher angegeben werden, *was* die Methode macht, als *wie* sie es macht. Ebenso sollten Parameter und Rückgabewerte angegeben werden. Dazu werden sogenannte »Tags« benutzt, mit denen eine Vielzahl von Informationen innerhalb eines Dokumentationskommentars an »Javadoc« übergeben werden kann. Eine Übersicht der wichtigsten Tags liefert Tabelle 3.1.

Tag	Beschreibung	Verwendung
`@param`	Beschreibung eines Parameters	`@param Parametername Beschreibung`
`@return`	Beschreibung des Rückgabewerts	`@return Beschreibung`
`@see`	Verweis auf ein anderes Packet/Methode/Attribut	`@see Methodenname`
`@exception/ @throws`	Angabe der Exception, die die Methode werfen kann	`@exception Exceptionname`
`@author`	Angabe des Autors	`@author Autorenname`
`@deprecated`	Gibt an, dass die Methode veraltet ist und nicht mehr verwendet werden sollte	`@deprecated Beschreibung`

Tabelle 3.1.: Javadoc Tags

In Eclipse kann eine Javadoc-Dokumentation über File → Export und Auswahl von »Javadoc« in der Kategorie »Java« angelegt werden. Um die Verwendung einer solchen Dokumentation zu verdeutlichen, wurde Beispielprogramm 3.4 vollständig mit Javadoc-Kommentaren versehen:

Programm 3.5: ExampleRobot mit Konstruktor (Javadoc)

```
1  /**
2  * Example Robot-class.
3  * Object is being created in main-method
4  * and variables are initialized with a non default-
       constructor.
5  */
6  public class ExampleRobot {
7      /**
8       * Name of the robot
9       */
10     public String name;
11     /**
12      * Price of the robot
13      */
14     public double price;
15     /**
16      * Year the robot was bought in
17      */
18     public int yearOfPurchase;
19     /**
20      * Appearance of the robot
21      */
22     public String appearance;
23     /**
24      * Returns the age of the robot.
25      * @param currentYear The current year
26      * @return The age of the robot
27      */
28     public int age(int currentYear) {
29         return currentYear - yearOfPurchase;
30     }
31     /**
32      * Constructor, that initializes the instance-
            variables with given parameters
33      * @param name Name of the robot
34      * @param price Price of the robot
35      * @param yearOfPurchase Year the robot was bought in
36      * @param appearance Appearance of the robot
37      */
38     ExampleRobot(String name, double price, int
           yearOfPurchase, String appearance) {
39         this.name = name;
40         this.price = price;
41         this.yearOfPurchase = yearOfPurchase;
42         this.appearance = appearance;
43     }
44     /**
45      * main-method. Object is created here.
46      * @param args standard parameter for main-method
47      */
48     public static void main(String[] args) {
49         ExampleRobot myEV3;
```

```
50          myEV3 = new ExampleRobot("LEGO EV3", 319.95,
                2013, "Roberta");
51      }
52  }
```

Die resultierende Javadoc-Dokumentation kann unter www.roberta-home.de/javaband-ev3/examplerobot-doc.html angesehen werden.

3.4 Variablen und Methoden

Variablen

Die grundsätzliche Funktionsweise von Variablen unterscheidet sich nicht von der anderer Programmiersprachen. Eine Übersicht über die Datentypen, mit denen Variablen belegt werden können und die leJOS bereitstellt, gibt Tabelle 3.2 auf Seite 52. Im Folgenden soll der Unterschied zwischen

- Instanzvariablen
- Klassenvariablen
- lokalen Variablen

deutlich werden.

Abhängig von der Art, wie eine Variable deklariert wird, ist sie entweder eine Instanz- (instance-), eine Klassen- (class-) oder eine lokale Variable (local variable). Eine *Instanzvariable* ist, wie der Name andeutet, eine Variable, die nur dem gemeinsam mit ihr erzeugten Objekt zur Verfügung steht. Eine *Klassenvariable* hingegen steht allen Objekten einer Klasse zur Verfügung. Werden also, wie in dem Beispiel beschrieben, einige Objekte von einer Klasse erzeugt, so erhalten sie die Eigenschaften aus dieser Klasse. Obwohl es in der Klasse nur ein Attribut `appearence` (Aussehen) gibt, kann jedes Objekt (also jeder konkrete Roboter) diesen Wert individuell beschreiben und verändern, ohne damit die anderen Objekte zu beeinflussen.

Soll aber zum Beispiel erfasst werden, wie viele Objekte der Klasse `ExampleRobot` erzeugt wurden, so benötigt die Klasse einen Zähler, auf den alle Objekte zugreifen können. Dieser Zähler ist dann eine *Klassenvariable*: Der Inhalt der Variable ist für alle Instanzen gleich. Eine solche Variable wird bei Deklaration durch das Schlüsselwor `static` gekennzeichnet.

Lokale Variablen werden innerhalb einer Methode definiert und existieren auch nur dort. Das Beispielprogramm 3.6 zeigt die verschiedenen Variablendeklaration und die Verwendung einer Klassenvariablen.

Programm 3.6: Variablendeklaration

```
1  /**
2  * Example Robot-class. Illustrates the definition of
       different variable-types.
3  */
4  public class ExampleRobot {
5      /**
6      * Class-variable for counting the total number of
           created objects.
7      */
8      private static int robotCount = 0;
9
10     //declaration of instance- variables
11     public String name;
12     public double price;
13     public int yearOfPurchase;
14     public String appearance;
15
16     /**
17     * Constructor that increases the class-variable
           robotCount by 1 upon object-creation.
18     */
19     ExampleRobot() {
20         robotCount = robotCount + 1;
21     }
22
23     public int age(int currentYear) {
24         //declaration of a local variable
25         int time = currentYear - yearOfPurchase;
26         return time;
27     }
28 }
```

Methoden

Methoden geben einer Klasse ihre Fähigkeiten und ihr Verhalten. Konkrete Aufgaben wie Berechnungen können übersichtlich in einzelne Methoden gefasst werden. Eine Methode hat immer einen Rückgabewert und optional auch Eingabeparameter, mit denen einer Methode Werte übergeben werden können.

Methoden sind entweder objekt- oder klassenbezogen. Klassenmethoden werden durch das Schlüsselwort `static` deklariert. Diese Klassenmethoden können innerhalb oder außerhalb der Klasse aufgerufen werden. Durch den sogenannten Modifier werden die Zugriffsregeln auf die Methode bestimmt.

Der Modifier `private` erlaubt nur den Zugriff von der eigenen Klasse. Durch den Modifier `public` können beliebige anderen Klassen auf die Methode zugreifen. Eine vertiefende Beschreibung der Modifier und ihrer Verwendung befindet sich unter dem Punkt Kapselung in Kapitel Kapitel 4 ab Seite 53.

Methoden können mit Übergabeparametern angelegt werden. Diese Parameter werden dann beim Aufruf direkt mit angegeben und stehen in Klammern nach dem Methodennamen. Der Rückgabewert der Methode wird mit `return` an den Aufruf zurückgegeben. Ein Beispiel für einen solchen Übergabeparameter haben wir bereits in der Klasse `ExampleRobot` gesehen. Der Methode `age` wird das aktuelle Jahr (`int currentYear`) als Integer übergeben und so mit Hilfe des Kaufjahres (`yearOfPurchase`) das Alter des Roboters berechnet. Dieser Wert wird dann mittels `return` zurückgegeben.

Das nachfolgende Beispielprogramm zeigt einige Methoden mit verschiedenen Modifiern, Übergabeparametern und Rückgabewerten. Zur anschaulichen Darstellung werden die Rückgabewerte auf dem Display des EV3 ausgegeben. Dabei werden vorab Methoden der Klassen `LCD` und `Button` verwendet, die eigentlich erst in Kapitel 7 ab Seite 81 beschrieben werden.

`LCD.drawString(...)`
: gibt einen String, der als erster Parameter übergeben wird, auf dem Display aus. Die beiden anderen Parameter geben die Stelle auf dem Display an, an der der Anfang des Strings erscheinen soll (Spalte, Zeilte).

`Button.ENTER.waitForPress()`
: wartet mit der Programmausführung solange, bis die ENTER-Taste (mittlere Taste auf dem EV3-Stein) gedrückt wurde. Diese Methode wird verwendet, damit die Ausgabe auf dem Display für einige Zeit bestehen bleibt. Bei Programmende kehrt der EV3 nämlich zum Hauptmenü zurück und die Ausgabe des Programms verschwindet.

Programm 3.7: MethodTester

```
1  import lejos.hardware.Button;
2  import lejos.hardware.lcd.LCD;
3
4  /**
5  * MethodTester-class.
6  * Implements different methods.
7  */
8  public class MethodTester {
9      /**
10     * 1. method: Returns a string
11     */
12     private String returnString() {
13         return "Example string";
14     }
15     /**
16     * 2. method: squares the given value
17     */
18     public static int square(int value) {
19         return value*value;
20     }
21     /**
22     * 3. method: implements "value" power "exponent"
23     */
24     public double power(double value, double exponent) {
25         return Math.pow(value,exponent);
26     }
27     /**
28     * 4. method: implements a*b*c
29     */
30     public int product(int a, int b, int c) {
31         return a*b*c;
32     }
33
34     public static void main (String args[]) {
35         MethodTester methods = new MethodTester();
36
37         LCD.drawString("1. "+methods.returnString(),0,0);
38         LCD.drawString("2. "+MethodTester.square(15),0,1)
               ;
39         LCD.drawString("3. "+methods.power(3.5,7),0,2);
40         LCD.drawString("4. "+methods.product(3,4,5),0,3);
41
42         Button.ENTER.waitForPress();
43     }
44 }
```

Das Programm 3.7 »MethodTester.java« führt zu folgender Ausgabe auf dem EV3-Display:

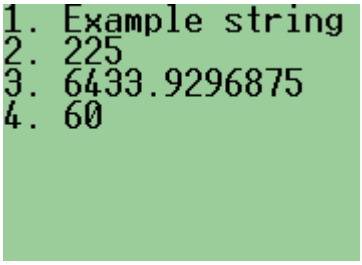


Abbildung 3.1.: Ausgabe des MethodTester-Programms

3.5 Ausdrücke, Anweisungen und Blöcke

Ausdrücke

Ausdrücke liefern bei ihrer Auswertung ein Ergebnis zurück. Sie bestehen aus Variablen, Operatoren und Methodenaufrufen. Einfache Beispiele geben die arithmetischen Operatoren (»+«, »–«, »*«, »/«), die zusammen mit zwei Operanden einen Ausdruck bilden. Der Datentyp des Ergebnisses hängt vom Typ der verwendeten Operanden ab.

Der »+«-Operator hat in Java neben der arithmetischen Verknüpfung zweier Zahlen noch eine weitere sehr praktische Funktion: Das Zusammenfügen von zwei Strings. Der Ausdruck "`Ich bin ein "` + "`String`" liefert damit den String "`Ich bin ein String`" zurück. Dabei konvertiert der »+«-Operator auch Zahlenwerte aus Variablen zu Strings. Der Ausdruck "`Der Wert der Variable intValue lautet: "` + `intValue` liefert einen String zurück, der den aktuellen Wert der Variable intValue angibt.

Auch die logischen Operatoren »==« (gleich), »!=« (ungleich), »>« (größer), »<« (kleiner), »>=« (größer-gleich) und »<=« (kleiner-gleich) bilden mit zwei Operanden einen Ausdruck. Bei Auswertung liefert dieser einen booleschen Wert (`true` oder `false`) zurück, der den Wahrheitsgehalt des Ausdrucks angibt.

Anweisungen

Anweisungen bestehen aus Ausdrücken und bilden die kleinste ausführbare Einheit eines Computerprogramms. Es gibt drei verschiedene Arten von Anweisungen.

Ausdrucksanweisungen sind Ausdrücke die lediglich durch Abschluss mit einem Semikolon zu einer Anweisung werden. Dazu gehören beispielsweise Funktionsaufrufe oder Wertzuweisungen. Bei einer Wertzuweisung mit dem »=«-Operator wird der Ausdruck `intValue = 3` durch Abschluss mit einem Semikolon zu einer Anweisung. Es handelt

sich dabei ohne Semikolon um einen Ausdruck, da der Wert auch zurückgeliefert wird.

Weiter gibt es Kontrollflussanweisungen, die in Abschnitt 3.6 beschrieben werden und Deklarationsanweisungen. Bei diesen handelt es sich um jede Art von Variablendeklarationen, also zum Beispiel: `int intValue;`. Dabei wird dem Programm die Existenz einer Variable bekannt gemacht, bevor mit ihr gearbeitet wird. Deklaration und Zuweisung können auch in einer Anweisung kombiniert werden: `int intValue = 0;`.

Blöcke

Ein Block ist eine Gruppierung von mehreren Anweisung mit Hilfe geschweifter Klammern. Der Beginn eines Blocks wird durch »{«, das Ende eines Blocks durch »}« gekennzeichnet. Ein Block kann überall da verwendet werden, wo auch einzelne Anweisungen erlaubt sind. Ein Beispiel findet sich in der Syntax der if-else-Anweisung auf Seite 46.

3.6 Kontrollstrukturen

Kontrollstrukturen sind Anweisungen, die den Programmfluss steuern, also die Reihenfolge beeinflussen, in der Anweisungen ausgeführt werden. Eine Kontrollstruktur ist entweder eine Verzweigung (if-else und switch) oder eine Schleife (while, do-while und for).

if-else-Anweisung

Die `if`-Struktur wird verwendet, um Programmverzweigung aufgrund einer Bedingung zu realisieren. Nur wenn die Bedingung zutrifft, werden die Anweisungen nach dem Schlüsselwort `if` abgearbeitet, ansonsten werden die Anweisungen nach `else` abgearbeitet. Das folgende Flussdiagramm in Abbildung 3.2 erläutert die allgemeine Programmverzweigung der if-Struktur.

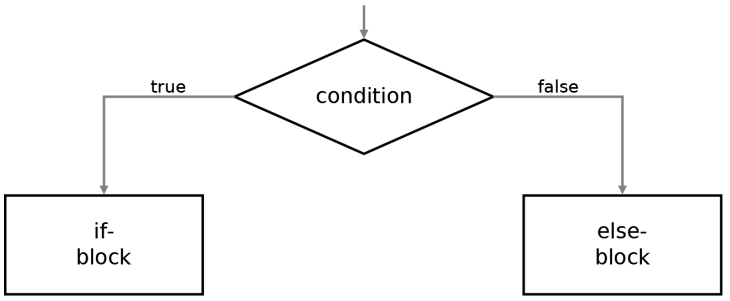


Abbildung 3.2.: Flussdiagramm der if-Struktur

Die Java-Syntax zur Implementierung einer `if`-Abfrage lautet:

```
if (condition) {
    statement;
    ⋮
}
else {
    statement;
    ⋮
}
```

Mehrere `if` bzw. `if-else`-Anweisungen können beliebig verschachtelt werden. Hierzu wird eine zusätzliche `if`-Anweisung in den Anweisungsblock von `else` geschrieben. Es ist natürlich auch möglich, nur die `if`-Anweisung (ohne den `else`-Teil) zu nutzen.

switch-Anweisung

Im Gegensatz zur Zweifachverzweigung der einfachen `if-else`-Struktur, können mit dem `switch`-statement Mehrfachverzweigungen im Programmfluss erstellt werden.

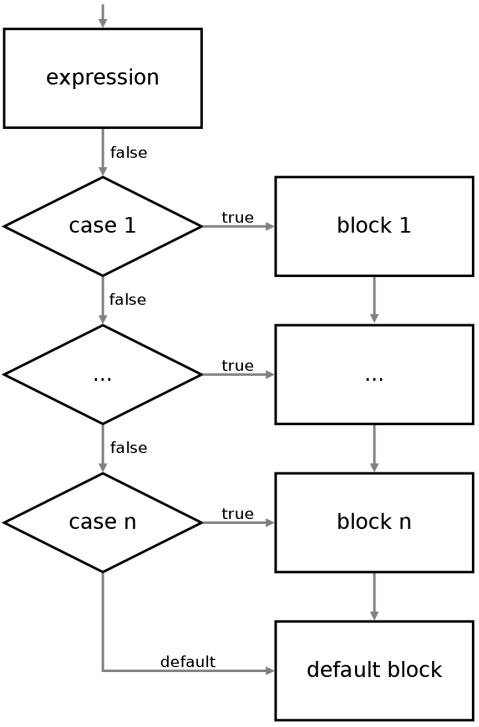


Abbildung 3.3.: Flussdiagramm des switch-case-statements

Dazu wird zunächst der Wert einer Variablen oder eines Ausdrucks evaluiert und je nach Ergebnis unterschiedliche Anweisungen ausgeführt. Die Syntax des `switch`-statements lautet wie folgt:

```
switch (expression){
    case result1:
        statement;
        ⋮
    case result2:
        statement;
        ⋮
    ⋮
    case resultN:
        statement;
        ⋮
    default:
        statement;
        ⋮
}
```

Der zu evaluierende Ausdruck (meist eine Variable) wird in Klammern hinter dem Schlüsselwort `switch` angegeben. Im darauffolgenden Switch-Block können mittels `case` für beliebig viele Ergebnisse dieses Ausdrucks unterschiedliche Anweisungen angegeben werden. Ist ein Ergebnis im Switch-Block nicht zu finden, werden die Anweisungen hinter `default` ausgeführt.

Wie in Diagramm 3.3 zu erkennen ist, werden – falls nicht anders behandelt – für ein bestimmtes Ergebnis nicht nur die dafür angegebenen Anweisungen ausgeführt, sondern auch **alle** darauf folgenden Anweisungen anderer Ergebnisse. Das liegt daran, dass mit `case` und `default` lediglich Sprungziele im Switch-Block für den Programmfluss definiert werden. Nach einem Sprung zur dem Ergebnis entsprechenden Stelle wird der Switch-Block dann regulär Zeile für Zeile abgearbeitet.

In der Regel sollen die Anweisungen anderer Ergebnisse nicht mit ausgeführt werden. Daher muss der Switch-Block nach jeder Sequenz von Anweisungen für ein Ergebnis explizit mittels `break` verlassen werden.

Dieses Verhalten des `switch`-Statements ist durchaus gewünscht und kein Fehler. So ist es nämlich möglich, durch Weglassen des `break`, für mehrere Ergebnisse die gleichen Anweisungen auszuführen. Man spricht dann auch vom sogenannten »fall through«.

while-Schleife

Die `while`-Schleife wird verwendet, um Wiederholungen von bestimmten Anweisungen auszuführen, solange (engl. *while*) eine Bedingung erfüllt ist (siehe Abbildung 3.4). Die `while`–Schleife wird auch kopfgesteuerte Schleife genannt. Es wird zunächst die Bedingung im Schleifenkopf geprüft, ist diese erfüllt (`true`), werden die Anweisungen im Schleifenrumpf ausgeführt. Dies wiederholt sich, bis die Bedingung nach einer Ausführung des Schleifenrumpfs nicht mehr erfüllt ist.

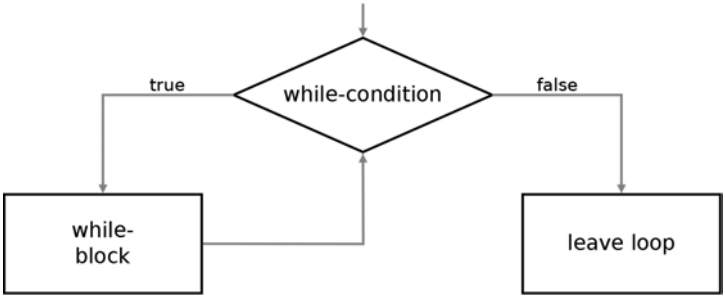


Abbildung 3.4.: Flussdiagramm der while-Schleife

Die Java-Syntax zur Implementierung einer solchen `while`-Schleife lautet wie folgt:

```
while (condition) {
    statement;
    ⋮
}
```

do-while-Schleife

Soll der Anweisungsblock mindestens einmal ausgeführt werden, empfiehlt sich die Verwendung der `do-while`-Schleife. Bei ihr wird im Programmverlauf zuerst die Anweisung ausgeführt und danach geprüft, ob die Bedingung noch gültig ist. Daher wird die `do-while`–Schleife als fußgesteuerte Schleife bezeichnet.

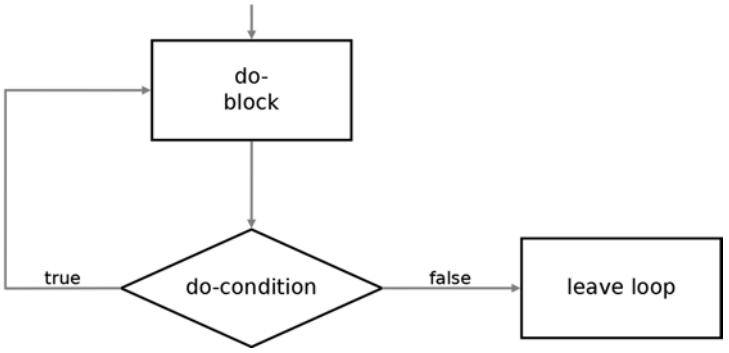


Abbildung 3.5.: Flussdiagramm der do-while-Schleife

Die Java-Syntax zur Implementierung einer `do-while`-Schleife lautet::

```
do {
    statement;
    ⋮
} while (condition)
```

for-Schleife

Braucht man in einer wiederholenden Struktur einen Schleifenzähler, empfiehlt es sich, statt der `while`-Schleife eine `for`-Schleife zu verwenden.

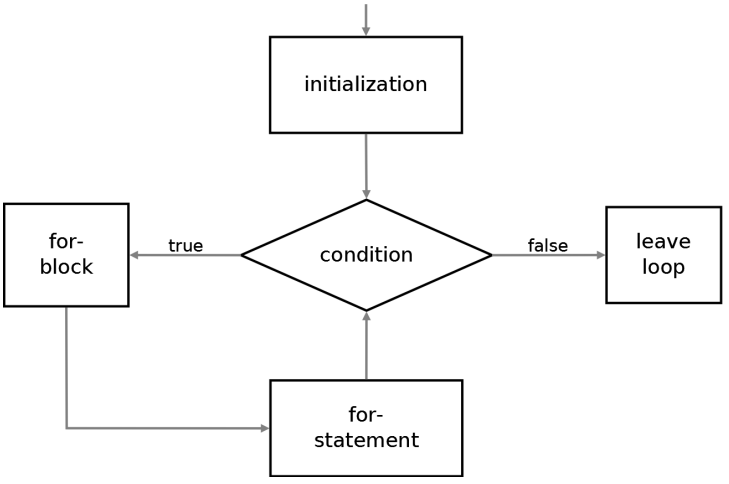


Abbildung 3.6.: Flussdiagramm der for-while-Schleife

Die Initialisierung (engl. initialization) wird ausgeführt bevor die `for`-Schleife das erste Mal durchlaufen wird. An dieser Stelle wird die Zählvariable erstellt. Danach folgt die Bedingung (engl. condition), die vor jedem Durchlauf der Schleife geprüft wird. Liefert sie `true` wird der nächste Durchlauf gestartet. Nach jedem Durchlauf wird der sogenannte Fortschaltausdruck (for statement) ausgeführt. Dieser verändert die Zählvariable auf gewünschte Art und Weise, kann aber theoretisch jede beliebige Anweisung enthalten.

Die `for`-Struktur sieht wie folgt aus:

```
for(initialization; condition; statement){
    statement;
        ⋮
}
```

Eine mögliche `for`-Schleife, im Laufe derer die Zählvariable von 0 bis 8 erhöht wird, sieht dann beispielsweise so aus:

```
for(int i = 0; i <= 8; i++){
    statement;
        ⋮
}
```

Die Integer-Variable i wird zu Beginn mit dem Wert `0` initialisiert. Die Bedingung `i<=8` bedeutet, dass die Anweisungen solange wiederholt werden, wie `i` kleiner oder gleich `8` ist. Der Ausdruck `i++` (gleichbedeutend mit `i=i+1`) erhöht den Wert der Zählvariable nach jeder Ausführung des `for`-Blocks um `1`. Alternativ können so auch Schleifen konstruiert werden, deren Zählvariablen kleiner werden.

Schleifensteuerung

Manchmal ist es nötig eine Schleife vorzeitig zu beenden oder eine Schleifeniteration an einer bestimmten Stelle abzubrechen und mit der nächsten Iteration zu beginnen. Dazu stehen dem Programmierer drei Befehle zur Verfügung:

`break` beendet die Schleife komplett. Die Programmausführung wird nach dem Schleifenrumpf, also ausserhalb der Schleife, fortgesetzt. Meist wird dieser Befehl innerhalb einer Schleife in Verbindung mit `if`-Abfragen eingesetzt, um eine oder mehrere Abbruchbedingungen für die Schleife zu erstellen.

`continue` beendet die aktuelle Schleifeniteration und setzt die Programmausführung im Schleifenkopf (bzw. Schleifenfuß bei `do-while`-Schleifen) fort. Bei `while`- und `do-while`-Schleifen wird mit dem Testen der Schleifenbedingung fortgefah-

ren. Bei `for`-Schleifen wird zuerst der Fortschaltausdruck ausgewertet und dann die Schleifenbedingung getestet. Das Programm springt daraufhin bei allen drei Schleifen (falls die Schleifenbedingungen wahr sind) wieder in den Schleifenrumpf.

`return` haben wir bereits kennengelernt. Schleifen können damit verlassen werden, wenn sie sich innerhalb von Methoden befinden, die nicht `void` sind, also einen Rückgabewert liefern müssen. Es wird der hinter `return` angegebene Wert an den Methodenaufruf zurückgeliefert, die Methode dadurch beendet und damit auch die Schleife innerhalb der Methode abgebrochen.

Zu beachten ist, dass sich `break` und `continue` nur auf den Block beziehen in dem sie aufgerufen werden. Bei verschachtelten Schleifenstrukturen laufen die äußeren Schleifen unverändert weiter, wenn in der Innersten ein `break` oder `continue` aufgerufen wird.

3.7 Datentypen

leJOS unterstützt alle Datentypen, die auch von Java unterstützt werden. In Tabelle 3.2 auf der nächsten Seite sind die wichtigsten von Java unterstützten Datentypen aufgeführt. Aus Performancegründen sind die primitiven Datentypen (char, byte, short, int, long) nicht in echten Klassen definiert, sondern in sogenannten Wrapper-Klassen. Wrapper-Klassen sind Klassen, die den entsprechenden Datentyp in ein Objekt »einpacken«. Hinzu kommt, dass es nicht, wie in anderen Programmiersprachen, `unsigned`-Typen gibt. Positive und negative Wertebereiche werden implizit deklariert.

Art	Name/ Klasse	Datentyp	Größe	Wertebereich
Binär	Boolean	boolean	1 Bit	true oder false
Ganzzahl	Byte	byte	8 Bit	-2^7 bis -2^7-1
Ganzzahl	Short	short	16 Bit	-2^{15} bis $2^{15}-1$
Ganzzahl	Integer	int	32 Bit	-2^{31} bis $2^{31}-1$
Ganzzahl	Long	long	64 Bit	-2^{63} bis $2^{63}-1$
Fließkommazahl	Float	float	32 Bit	$\pm 1,4E-45$ …$\pm 3,4E+38$
Fließkommazahl	Double	double	64 Bit	$\pm 4,9E-324$ …$\pm 1,7E+308$
Zeichen	Char	char	16 Bit	
Zeichen	String	String	max. $2^{31}-1$ Zeichen	

Tabelle 3.2.: Wertebereiche von Datentypen

Konzepte der OOP

4.1 Generalisierung

Bei der Generalisierung wird ausgenutzt, dass unterschiedliche Klassen teilweise gleiche Eigenschaften haben. Diese Eigenschaften müssen nur ein Mal beschrieben werden und können dann von allen erbenden Klassen genutzt werden. Ein EV3-Roboter und ein NXT-Roboter sind zwar sehr verschieden in Hinsicht auf die verbauten Komponenten, jedoch können diese beiden Elemente eindeutig als Roboter klassifiziert werden. Beide Roboter verfügen über vergleichbare Attribute (Eigenschaften) und Methoden, denn jeder EV3-Roboter und jeder NXT-Roboter hat beispielsweise eine Bezeichnung, eine Anzahl Motoren und eine bestimmte Geschwindigkeit, mit der die Motoren arbeiten sollen.

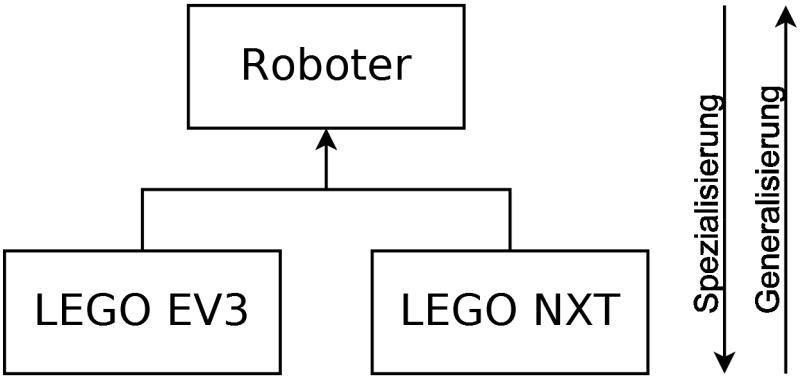


Abbildung 4.1.: Spezialisierung und Generalisierung

Fähigkeiten, die sowohl ein EV3 als auch ein NXT hat, können dann in der sogenannten Oberklassse `Roboter` implementiert werden. Ein Beispiel hierfür ist die Fähigkeit, die Geschwindigkeit der Motoren zu verändern. Wird diese Fähigkeit (Methode) in der Klasse Roboter beschrieben, gilt sie gleichermaßen für den EV3 und den NXT und braucht dort nicht erneut implementiert zu werden. Anstehende Änderungen müssen also nur an einer Stelle vorgenommen werden. Aus Sicht einer Oberklasse werden die Klassen, die in der Vererbungshierarchie weiter

unten liegen, immer konkreter. Dies wird Spezialisierung genannt. In umgekehrter Richtung ist von Generalisierung die Rede (siehe Abbildung 4.1 auf der vorherigen Seite).

UML Klassen können übersichtlich durch UML-Diagramme dargestellt werden. Die Unified Modeling Language (Vereinheitlichte Modellierungssprache), kurz UML, ist eine grafische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Software-Teilen und anderen Systemen. Sie wird von der Object Management Group (OMG) entwickelt und ist sowohl von ihr als auch von der ISO (ISO/IEC 19505 für Version 2.1.2) standardisiert (Wikipedia 2014). Zusätzliche Informationen finden sich unter:

- http://www.uml.org/
- http://www.oose.de/nuetzliches/fachliches/uml/

Abbildung 4.2 zeigt ein einfaches Beispiel für ein UML-Klassendiagramm: Es besteht aus drei übereinander liegenden Teilen. Im obersten Teil wird der Name der Klasse dargestellt. Zur besseren Übersicht wird er fett gedruckt. In der Mitte sind die Attribute dargestellt. Dies sind alle Variablen, die die Klasse bereitstellt. Im unteren Teil stehen die Methoden mit ihren Übergabeparametern und Rückgabewerten.

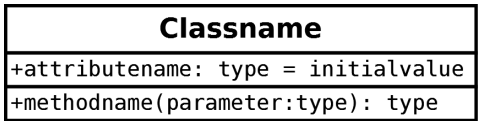


Abbildung 4.2.: UML-Klassendiagramm

4.2 Vererbung

Wir wollen nun das Prinzip der Vererbung in der objektorientierten Programmierung anhand eines Beispiels erläutern. Aus einer Oberklasse `Robot` werden zwei konkrete Unterklassen (`Ev3` und `Nxt`) abgeleitet (siehe Abbildung 4.3 auf der nächsten Seite). Die Unterklassen erben alle Methoden und Eigenschaften der Oberklasse. Dafür wird in Java das Schlüsselwort `extends` verwendet.

```
class EV3 extends Robot
```

In der Oberklasse Robot existieren die Einträge `name` für die Bezeichnung des Roboters und `motorSpeed` für die Geschwindigkeit, mit der die verbauten Motoren arbeiten. Zusätzlich gibt es den Eintrag `motorCount` (Motorenanzahl), der mit dem Wert 0 initialisiert wird. Diese Werte werden an Ev3 und Nxt vererbt, wo sie dann konkret belegt werden.

Neben den größeren Motoren können Roboter, die auf dem EV3-System basieren, auch kleinere Motoren verwenden. In unserem Beispiel soll der EV3-Roboter mit so einem Motor eine Greiferklaue öffnen und schließen können. Dafür wird die Eigenschaft `smallMotorCount` (Anzahl kleiner Motoren) und die Methoden `openClaw` sowie `closeClaw` in `EV3` implementiert. Da diese Eigenschaften und Methoden nicht aus der Oberklasse geerbt wurden, stehen sie auch nur der Klasse `EV3` zur Verfügung. Genau so wird in der Klasse `NXT` die Eigenschaft `currentArmAngle` sowie die Methode `moveArmTo` implementiert, die der NXT-Roboter benötigt, um einen Greifarm zu schwenken und die dann auch nur der Klasse `NXT` zur Verfügung stehen.

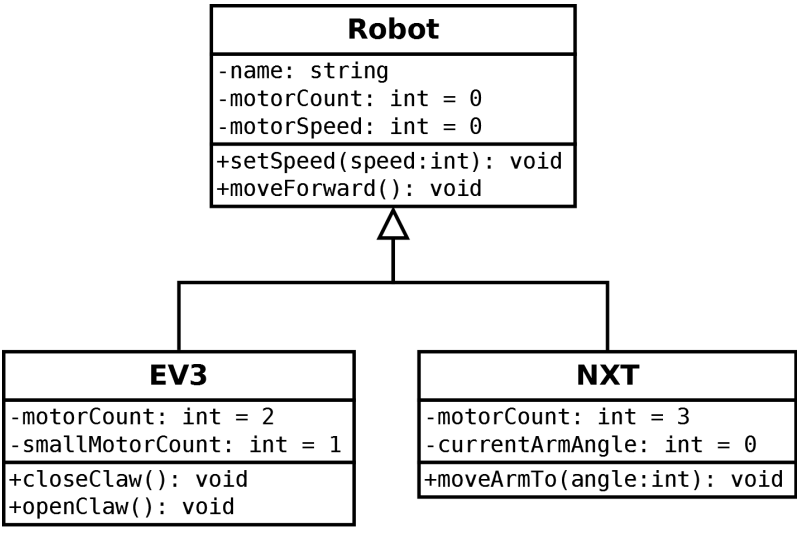


Abbildung 4.3.: UML-Diagramm: Vererbung

*In Java ist es **nicht** möglich, dass eine Klasse von mehreren Klassen erbt.*

Dieses Vererbungsprinzip erlaubt es, dass Vererbungen beliebig geschachtelt werden können. Die abgeleitete Klasse erbt die Eigenschaften der Oberklasse, die wiederum die Eigenschaften ihrer Oberklasse erbt usw.

4.3 Kapselung

Die Kapselung von Informationen wird auch als »data hiding« bezeichnet. Damit werden die Eigenschaften innerhalb einer Klasse »versteckt«, um Zugriffe von außen zu verhindern. Den Variablen wird hierzu bei ihrer Deklaration eine Zugriffsberechtigung zugewiesen. Dies geschieht über eines der vier Schlüsselwörter (Modifier)

- `public` (öffentlich),
- `private` (privat) und
- `protected` (geschützt).

Wird einer Variablen keine Zugriffsberechtigung zugewiesen, so erhält diese den Zustand `default` (siehe Tabelle 4.1).

	public	protected	default	private
Selbe Klasse	Ja	Ja	Ja	Ja
Selbes Package	Ja	Ja	Ja	Nein
Unterklasse (extends)	Ja	Ja	Nein	Nein
Überall (andere Klassen etc.)	Ja	Nein	Nein	Nein

Tabelle 4.1.: Zugriffsrechte

Java-Programme bestehen aus Ansammlungen von vielen einzelnen Dateien, die innerhalb der objektorientierten Programmierung jeweils eine Klasse repräsentieren. Um dieser Klassensammlung eine Struktur geben zu können und somit die Les- und Benutzbarkeit für Menschen zu verbessern, wurden Java Packages eingeführt. (ITWissen o.D.)

In UML-Notation wird dies folgendermaßen umgesetzt:

+ public (öffentlich)

– private (privat)

protected (geschützt)

~ (Paketsichtbar)

Es wird empfohlen, sich für jede Klasse zu überlegen, welche Variablen und Methoden von außen angesprochen werden sollen und welche nur innerhalb der Klasse oder des Package zur Verfügung stehen müssen.

Allgemein gilt: Was nicht für die Verwendung von außen, sondern nur für die Programmierung im Inneren notwendig ist, sollte im Inneren verborgen werden. Um ungültige Datenwerte in den Datenfeldern (Variablen, Arrays etc.) zu vermeiden, ist es empfehlenswert, alle Datenfelder als `private` oder `protected` zu definieren und eigene von außen ansprechbare Methoden als `public` für das Setzen und Abfragen der Werte vorzusehen. Dies ist das Konzept der Getter- und Setter-Methoden. Per Konvention sollen diese Methoden Namen der Form setXxxx und getXxxx haben. Eine solche Methode haben wir bewusst oder unbewusst schon in Abbildung 4.3 auf Seite 55 gesehen. Die Methode `setSpeed` wird als `public` deklariert, um von außen auf die `private`-Variable `motorSpeed` zugreifen zu können. Im Folgenden Codebeispiel wird die Oberklasse `Robot` mit motorSpeed-Setter implementiert und um einen motorSpeed-Getter erweitert.

Programm 4.1: Implementierung von Getter- und Setter- Methoden

```
1  public class Robot{
2
3      private String name;
4      private int motorCount = 0;
5      private int speed = 0;
6
7      /**
8      * Motorspeed-setter. Sets speed.
9      */
10     public void setSpeed(int speed) {
11         this.speed = speed;
12     }
13     /**
14     * Motorspeed-getter. Returns speed.
15     */
16     public int getSpeed() {
17         return speed;
18     }
19
20     public void moveForward() {
21         //add implementation of moveForward- method here
22     }
23 }
```

Dieses Konzept bietet den großen Vorteil, dass die Klassen über wohl definierte Schnittstellen (hier Methoden) angesprochen werden können. Änderungen in der Variablenstruktur müssen nur in einer Klasse umgesetzt werden. Die andere Klasse greift ausschließlich auf die Schnittstellenmethoden zu.

Ein weiterer Vorteil ist, dass diese Methoden auch der Kontrolle dienen können. Es kann beispielsweise eine logische Prüfung der übergebenen Werte stattfinden. So kann überprüft werden, ob der Wert für die Reifenzahl positiv und damit gültig ist. Ein negativer Wert bei der Angabe der Reifenanzahl eines Autos könnte auf diese Weise unterbunden werden.

this-Pointer

In der Setter-Methode wurde der sogenannte »this«-Pointer zum Setzen der Variable verwendet. Dieser stellt eine Referenz auf die Instanz dar, aus der die Methode aufgerufen wurde. Er wird an dieser Stelle benötigt, da die Instanzvariable `speed` sonst nicht vom Übergabeparameter `speed` zu unterscheiden wäre. Getter- und Setter-Methoden können in dieser Form auch automatisch von Eclipse durch Rechtsklick auf die gewünschte Variable und Auswählen von Source → Generate Getters and Setters... erstellt werden.

4.4 Polymorphie

Das Wort Polymorphie entstammt der griechischen Sprache und bedeutet »Vielgestaltigkeit«. Die Polymorphie der objektorientierten Programmierung ist eine Eigenschaft, die in Zusammenhang mit Vererbung einhergeht. Eine Methode ist genau dann polymorph, wenn sie von verschiedenen Klassen unterschiedlich genutzt wird.

Gibt es in einer Klassenhierarchie mehrere Methoden auf unterschiedlichen Hierarchieebenen, wird erst zur Laufzeit festgelegt, welche der Methoden für ein gegebenes Objekt verwendet wird. Bei einer mehrstufigen Vererbung wird jene Methode verwendet, die direkt in der Objektklasse definiert ist, oder jene, die im Vererbungszweig am weitesten »unten« liegt. Variablen für Objekte sind in Java stets polymorph. Das bedeutet, dass eine einzelne Variable vom Typ der Oberklasse für verschiedene Objekte erbender Klassen in einem Programm verwendet werden kann. Wenn die Variable verwendet wird, um eine Methode aufzurufen, hängt es vom Objekt ab, auf das die Variable gegenwärtig verweist, welche Methode ausgeführt wird. Das folgende Codebeispiel soll die Polymorphie in Java erläutern.

Programm 4.2: Polymorphie

```
1  import lejos.hardware.Button;
2  import lejos.hardware.lcd.LCD;
3
4  class Robot{
5      private String name;
6      private int motorCount = 0;
7      private int speed = 0;
8
9      public void setSpeed(int speed) {this.speed = speed;}
10     public int getSpeed() {return speed;}
11     public void moveForward() {
12         LCD.drawString("The ROBOT is moving", 0, 0);
13     }
14 }
15
16 class EV3 extends Robot{
17     private int motorCount = 2;
18     private int smallMotorCount = 1;
19
20     public void closeClaw() {//add implementation of
           closeClaw- method here}
21     public void openClaw() {//add implementation of
           openClaw- method here}
22     public void moveForward() {
23         LCD.drawString("The EV3 is moving", 0, 1);
24     }
25 }
26
27 class NXT extends Robot{
28     private int motorCount = 3;
29     private int currentArmAngle = 0;
30
31     public void moveArmTo(int angle) {//add
           implementation of moveArmTo- method here}
32
33     public void moveForward() {
34         LCD.drawString("The NXT is moving", 0, 2);
35     }
36 }
37
38 public class Polymorphie {
39     public static void main(String[] args) {
40         Robot anyRobot;
41         anyRobot = new EV3();
42         anyRobot.moveForward();
43
44         anyRobot = new NXT();
45         anyRobot.moveForward();
46
47         Button.ENTER.waitForPress();
48     }
49 }
```

Die Ausgabe dieses Programms lautet:

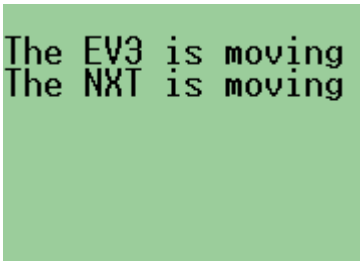


Abbildung 4.4.: Ausgabe auf dem Display des EV3

Die Methode `moveForward()` unterliegt also offensichtlich der Vielgestaltigkeit. Zweimaliges Aufrufen von der Variable `anyRobot` aus mit demselben Methodenaufruf führt zu unterschiedlichen Ausgaben. Dies geschieht, da die Methode in den Unterklassen `Ev3` und `Nxt` jeweils unterschiedlich überschrieben wurde.

Kapitel 5

Weiterführende Konzepte der OOP

Im Folgenden werden einige Konzepte vorgestellt, die einen entscheidenden Einfluss auf die Programmierung nehmen und komplexere Programmstrukturen ermöglichen.

5.1 Schnittstellen

Schnittstellen, auch als Interfaces bezeichnet, werden dazu verwendet, eine Trennung zwischen Spezifikation und Implementierung herzustellen.

Mit Schnittstellen kann bereits im Vorhinein festgelegt werden, welche Funktionalität von den implementierenden Klassen zur Verfügung gestellt werden soll. Bei der Konzeption einer Schnittstelle muss die Funktionsweise der verwendeten Methoden noch nicht bekannt sein. Schnittstellenklassen beinhalten ausschließlich abstrakte Methoden und Konstanten, die implizit öffentlich sind. Die Klassen selbst enthalten keine eigenen Variablen.

Anstatt des Schlüsselworts `class` wird eine Schnittstelle mit `interface` deklariert.

```
public interface DistanceSensor
```

Klassen, die eine solche Schnittstelle implementieren wollen, müssen das Schlüsselwort `implements` in ihren Header mit aufnehmen, gefolgt vom Namen der Schnittstelle.

```
public class SonarSensor implements DistanceSensor
```

Dies ermöglicht es, Objekte zu konstruieren, die eine gemeinsame Schnittstelle aufweisen, obwohl die Klassen nicht voneinander erben. Nützlich ist eine Schnittstellen-Klasse immer dann, wenn Eigenschaften einer Klasse überschrieben werden sollen, die nicht direkt in ihrer

normalen Vererbungshierarchie abgebildet werden können oder sollen. Soll zum Beispiel eine Verwaltung von Distanzsensoren entwickelt werden, so fällt schnell auf, dass alle eine `gibDistanz`-Methode benötigen. Je nach Sensor können diese Methoden unterschiedlich implementiert sein, was der Benutzer der Schnittstelle aber nicht zu wissen braucht.

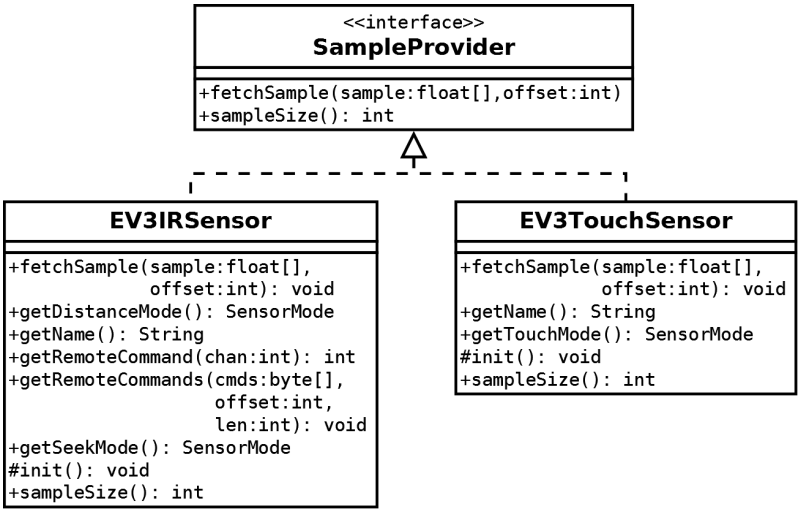


Abbildung 5.1.: UML-Diagramm einer Schnittstelle

Somit können die Module, die die gleiche Schnittstelle implementieren, ausgetauscht werden. Das Konzept der Schnittstelle dient also auch der Modularisierung. Das folgende Beispiel zeigt die Implementierung der oben genannten Schnittstelle und der Klassen, die diese Schnittstelle implementieren. In der benutzenden Klasse (FahreBisWand) könnte der SonarSensor durch LaserScanner ersetzt werden, ohne dass sich in der weiteren Funktionsweise etwas ändern würde. Den Benutzer der Schnittstelle brauchen die unterschiedlichen Implementierungen in den jeweiligen Klassen nicht zu interessieren.

Programm 5.1: Verwendung von Schnittstellen

```
1  /**
2  * Interface for distance measuring classes.
3  */
4  interface DistanceSensor {
5      public double readDistance();
6  }
7  /**
8  * This class describes a sonar sensor.
9  * readDistance() must be implemented here
10 */
11 class SonarSensor implements DistanceSensor {
12     public double readDistance() {
13         //specific implementation of readDistance()
14     }
15 }
16 /**
17 * This class describes a laser scanner.
18 * readDistance() must be implemented here as well
19 */
20 class LaserScanner implements DistanceSensor {
21     private int degree = 0;
22     public double readDistance() {
23         //specific implementation of readDistance()
24     }
25     public void changeDirection(int degrees) {
26         degree = degrees;
27     }
28 }
29
30 public class GoToWall {
31     public static void main(String[] args) {
32         DistanceSensor sensor;
33         sensor = new SonarSensor();
34         while (!(sensor.readDistance() < 3.0)) {
35             /*add implementation here
36             e.g. robot moves forwards*/
37         }
38         //and stops motors afterwards
39     }
40 }
```

5.2 Das Exception-Modell

Unter einer Ausnahme (engl. exception) versteht man in der Objektorientierten Programmierung das Auftreten bestimmter fehlerhafter Programmzustände. Bei ihrer Behandlung werden Informationen über diese Zustände strukturiert zur Laufzeit weitergegeben und gegebenenfalls an bestimmten Stellen im Programm darauf reagiert. Kann in einem

Programm z.B. eine Speicheranforderung nicht erfüllt werden, wird eine Speicheranforderungsausnahme ausgelöst. Mit Hilfe von *Exceptions* können somit Fehler ignoriert, behoben oder angezeigt werden.

Grundprinzip

Das Grundprinzip des *Exception*-Modells in Java lässt sich wie folgt beschreiben (Krüger, G. and Hansen, H. 2014):

- Ein Laufzeitfehler oder eine vom Entwickler gewollte Bedingung löst eine Ausnahme aus.
- Diese Ausnahme kann entweder vom Programmteil, in dem die Ausnahme ausgelöst wurde, behandelt werden oder sie kann weitergegeben werden.
- Wird die Ausnahme weitergegeben, so hat der Empfänger der Ausnahme erneut die Möglichkeit, sie entweder zu behandeln oder selbst weiterzugeben.
- Wird die Ausnahme von keinem Programmteil behandelt, so führt sie zum Abbruch des Programms und zur Ausgabe einer Fehlermeldung.

Als Beispiel für Ausnahmen kann man das Öffnen einer Datei heranziehen. Das Öffnen einer Datei kann aus verschiedenen Gründen fehlschlagen (keine Rechte, Datei nicht vorhanden etc.). Deshalb muss der Programmierer explizit angeben, was geschehen soll, wenn eine Datei nicht geöffnet werden kann (siehe Programm 5.3).

Ausgenommen hiervon sind Ausnahmetypen, die jederzeit auftreten können, wie zum Beispiel Indexfehler bei Indizierung eines Arrays. Diese sind meistens die Folge von Fehlern im Programm, sodass es hier nicht sinnvoll ist, die Ausnahme zu behandeln (stattdessen sollten derartige Fehler behoben werden).

Der try-catch-Block

Um Funktionen, die eine Ausnahme erzeugen benutzen zu können, müssen diese durch einen `try-catch`-Block umschlossen sein. Im `try`-Teil wird die Funktion, deren Ausführung gewünscht ist, eingetragen. Es wird also versucht, die Funktion aufzurufen. Schlägt der Aufruf fehl, wird eine Ausnahme ausgelöst, die dazu führt, dass die Anweisungen im `catch`-Block ausgeführt werden. Zusätzlich gibt es die Möglichkeit noch einen `finally`-Block einzubauen. Unabhängig davon, ob eine Ausnahme aufgetreten ist oder nicht, werden die Anweisungen hier immer ausgeführt. Der Block eignet sich dazu, »Aufräumarbeiten« durchzuführen.

Eine Exception wird beispielsweise auch geworfen, wenn probiert wird einen Sensor zu initialisieren, obwohl er nicht am angegebenen Sensorport angeschlossen ist.

Programm 5.2: Sensorinitialisierung

```
1  import lejos.hardware.port.SensorPort;
2  import lejos.hardware.sensor.EV3ColorSensor;
3
4  public class TryCatch {
5
6      public static void main(String args[]) {
7          EV3ColorSensor sensor = new EV3ColorSensor(
               SensorPort.S1);
8          /* further implementation */
9      }
10
11 }
```

Programm 5.2 führt ohne angeschlossenen Sensor dann zu folgender Ausgabe auf dem Display des EV3:

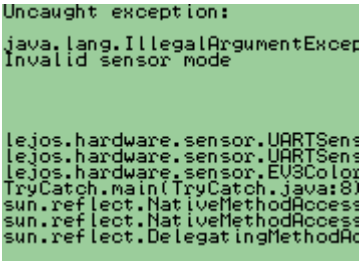


Abbildung 5.2.: IllegalArgumentException bei nicht angeschlossenem Sensor

Um den Anwender des Programms nicht mit dieser sperrigen (und für ihn vielleicht unaussagekräftigen) Ausgabe zu konfrontieren, kann ein `try-catch`-Block verwendet und damit auf die Exception reagiert werden:

Programm 5.3: Verwendung von Try-Catch

```
1  import lejos.hardware.Button;
2  import lejos.hardware.lcd.LCD;
3  import lejos.hardware.port.SensorPort;
4  import lejos.hardware.sensor.EV3ColorSensor;
5
6  public class TryCatch {
7
8      public static void main(String args[]) {
9
10         try{
11             EV3ColorSensor sensor = new EV3ColorSensor(
                   SensorPort.S1);
12             /* further implementation */
13         }
14         catch (IllegalArgumentException exc) {
15             LCD.drawString("Sensor missing!", 0, 0);
16             LCD.drawString("Plug it in and", 0, 1);
17             LCD.drawString("restart program.", 0, 2);
18             LCD.drawString("ESCAPE to exit", 0, 4);
19             Button.ESCAPE.waitForPress();
20         }
21
22     }
23 }
```

Dies führt dann zu folgender Ausgabe:

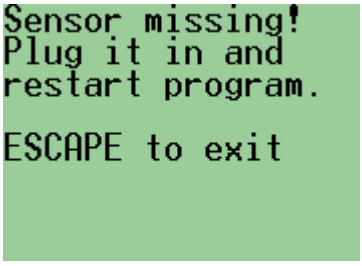


Abbildung 5.3.: Ausgabe des EV3 nach gefangener Exception

Ein nicht angeschlossener EV3-Berührungssensor wird keine Exception werfen, da es sich bei ihm um einen analogen Sensor handelt. Das Programm läuft fehlerfrei und der Sensor liefert bei jeder Abfrage eine »0« für »nicht gedrückt«.

Throws Exception

Neben dem Strukturieren von problematischen Blöcken durch einen `try-catch`-Block gibt es eine weitere Möglichkeit auf Exceptions zu reagieren. Hierzu wird im Methodenkopf die sogenannte `throws`-Klausel eingeführt.

- Throws mit Angabe der Exception:

```
throws IOException, FileNotFoundException
```

Im Folgenden zu sehen in einer Methode, die eine Zeile aus einer angegebenen Datei als String zurückgeben soll.

```
String readLineFromData(String data) throws IOException {
    RandomAccessFile d = new RandomAccessFile(data, "r");
    return d.readLine();
}
```

Durch `throws` gibt die Methode an, dass sie eine bestimmte Exception nicht selbst behandelt, sondern diese an die aufrufende Methode weitergibt. Ist es egal, wo die Fehlerbehandlung in einem Hauptprogramm behandelt wird, so können alle Fehler auch an die Laufzeitumgebung weitergeleitet werden, die dann das Programm im Fehlerfall abbricht. Dafür wird die `main`-Methode um eine *Exception* ergänzt.

- Throws ohne konkrete Angabe der Exception:

```
throws Exception
```

Programm 5.4: Verwendung von throws in der main-Methode

```
1 /**
2 * This class doesn't handle exceptions itself.
3 * It returns exceptions to runtime.
4 */
5 public class  ThrowsEverything {
6     public static void main(String args[]) throws
          Exception {
7         //Implementation
8     }
9 }
```

Im Gegensatz zur `try-catch`-Konstruktion, in der ganz konkret eine Ausnahmebehandlung für wenige Anweisungen implementiert wird, deutet `throws Exception` nur an, dass in dieser Methode generell

eine Ausnahme auftreten könnte. Somit wird der `try-catch`-Block überflüssig.

Im ersten Programm (es handelt sich um genau zu sein um einen Programmauszug, da auf die Klassenbezeichnung und die `main`-Methode verzichtet wurde) wird der `throws`-Befehl zusammen mit der Angabe einer konkreten Exception verwendet (hier: `FileNotFoundException`). Es lassen sich beliebig viele durch Kommata getrennte Exceptions angeben.

Das zweite Beispielprogramm verdeutlicht die Weitergabe einer Fehlerbehandlung an die Laufzeitumgebung (Die `main`-Methode wurde um `throws Exception` erweitert.)

5.3 Threads

Nebenläufigkeit

Threads schaffen in Java die Möglichkeit, Nebenläufigkeit innerhalb eines Programmablaufs zu erzeugen. Nebenläufigkeit bezeichnet die Fähigkeit, mehrere Vorgänge gleichzeitig ausführen zu können (siehe Abbildung 5.4).

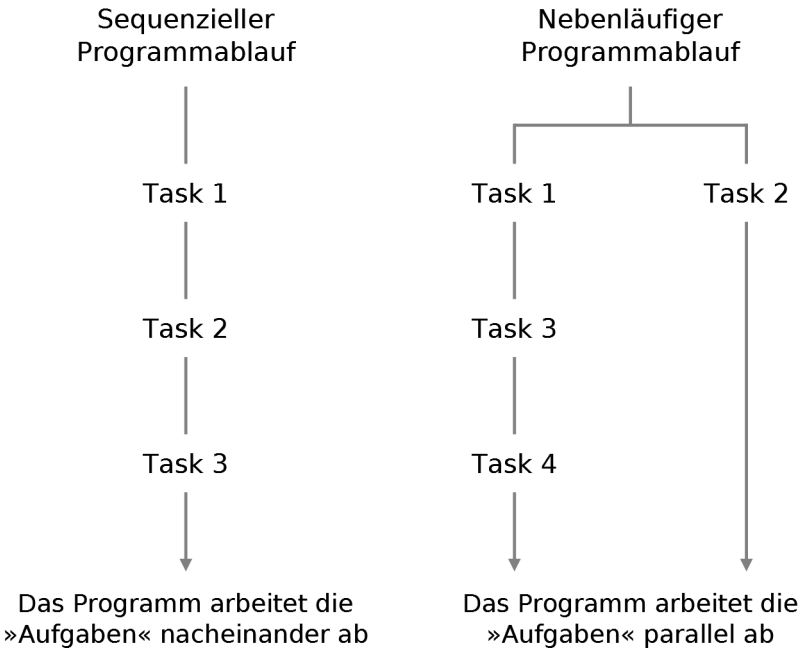


Abbildung 5.4.: Programmablauf mit und ohne Threads

Bei einem Programm mit sequentiellem Ablauf werden alle Programmteile nacheinander durchlaufen. Der Ablauf könnte mit einem Stift nachvollzogen werden. Dabei befindet sich die Spitze des Stiftes zunächst im Programmkopf, durchläuft danach Schleifen und Verzweigungen und landet schließlich am Ende des Programms. Nachteil: Soll ein Roboter während der Ausführung für fünf Sekunden vorwärts fahren, muss das Programm für diesen Zeitraum angehalten werden. In dieser Zeit kann es aber keine anderen Aufgaben mehr erledigen. Ein einzelner Pfad reicht also in manchen Fällen nicht aus. Stattdessen müssen mehrere Pfade gleichzeitig durch das Programm führen. So können beispielsweise auf einem Pfad die Motoren bewegt werden, während auf einem weiteren der Notstopp-Knopf ständig geprüft wird. Die einzelnen Pfad werden als »Thread« bezeichnet.

Nehmen wir einen autonomen Sicherheitsroboter, der nachts über ein Werksgelände fährt und Einbrecher mittels verschiedener Sensoren aufspüren soll. Hier müssen *Threads* verwendet werden, um die verschiedenen Aufgaben gleichzeitig ausführen zu können. In diesem Fall laufen die Abfragen der Sensoren in jeweils einem eigenen Thread für jeden Sensor, damit immer alle aktuellen Sensordaten zur Verfügung stehen und z.B. die Informationen der normalen Kamera nicht verloren gehen, wenn gerade die Infrarotkamera abgefragt wird. Außerdem muss sich der Roboter zeitgleich noch bewegen können, da er ja über das Gelände patrouillieren soll. Mit mehreren *Threads* ist es also möglich, verschiedene Aufgaben innerhalb eines Programms gleichzeitig zu erledigen.

Dabei wird ein *Thread* als Verwalter eingesetzt, der andere *Threads* starten oder beenden kann. Das Programm 5.5 auf Seite 70 besteht aus zwei Klassen. Die Klasse `Parent` übernimmt die Rolle des Verwalters. In der `main()`-Methode der Klasse `Parent`, wird zunächst ein neues Objekt mit dem Namen `child` vom Typ `Child` erzeugt. Die nächste Anweisung startet den *Thread*, indem es auf dem Objekt `child` die Methode `start()` aufruft. Die Methode `start()` ist in der Klasse `Thread` implementiert, von der die Klasse `Child` erbt.

Nach dem Aufruf von `start()` wird die Ausführung an die in der Klasse `Child` implementierte Methode `run()` übergeben. Von diesem Zeitpunkt an laufen die beiden *Threads* (Eltern- und Kindthread) gleichzeitig und können parallel Aufgaben erledigen.

Typischerweise ist der Elternthread für die Mensch-Maschine-Interaktion auf der einfachsten Ebene (zum Beispiel beim Notstopp) verantwortlich, während andere *Threads* konkrete Aufgaben im

Anwendungsbereich des Systems haben. Dies rührt daher, dass die Eingabe des Menschen höchste Priorität hat.

Im Programm 5.5 wartet der Elternthread nach dem Start des Kindthreads, bis die ESCPAE-Taste gedrückt wird, woraufhin das Programm beendet werden soll. In der Zwischenzeit sollen Im Kindthread die Werte eines Farbsensor zyklisch mit einer Sekunde Abstand ausgelesen und auf dem Display des EV3 ausgegeben werden. Dazu wird eine `while`-Schleife verwendet, die mittels `Thread.currentThread().isInterrupted()` den Zustand einer internen booleschen-Variable der `Child`-Klasse abfragt. Diese liefert nur dann `true`, falls zuvor die Methode `child.interrupt()` aufgerufen wurde (was nach Drücken der ESCAPE-Taste eintritt). Damit ist die Abbruchbedingung der `while`-Schleife erfüllt, diese wird beendet und die `run()`-Methode ist komplett durchlaufen. Das beendet den Kindthread und in diesem Fall auch das gesamte Programm.

Da die Farbwerte im Abstand von einer Sekunde ausgelesen werden sollen, muss der Kindthread nach jedem Durchlauf der `while()`-Schleife für eine Sekunde mittels `Thread.sleep(1000)` pausiert werden. Wenn nun aber in genau dieser Zeit die ESCAPE-Taste betätigt und damit `child.interrupt()` aufgerufen wird, muss der Kindthread seine Pause frühzeitig beenden, was `Thread.sleep(1000)` durch Werfen einer *InterruptedException* anzeigt. Diese Exception muss wie weiter oben beschrieben mit Hilfe eines `try-catch`-Blocks gefangen und behandelt werden. Im Beispiel wird die `while`-Schleife mittels `break` beendet und nochmals `child.interrupt()` aufgerufen, um die interne interrupted-Variable auf `true` zu setzen.

Programm 5.5: Eltern- und Child- Thread

```
1  import lejos.hardware.Button;
2  import lejos.hardware.lcd.LCD;
3  import lejos.hardware.port.SensorPort;
4  import lejos.hardware.sensor.EV3ColorSensor;
5  /**
6  * This class starts a thread and interrupts it after
       Button ENTER is pressed.
7  */
8  public class Parent {
9      public static void main(String args[]) throws
           InterruptedException {
10         Child child = new Child();
11         //start child-thread
12         child.start();
13         //wait until button ESCAPE is pressed
14         Button.ESCAPE.waitForPress();
```

```
15          child.interrupt();
16      }
17  }
18  /**
19  * This class represents a Thread.
20  * It reads color-values and displays them as long as it
        isn't interrupted.
21  */
22  class Child extends Thread {
23
24      public void run() {
25          //declaration and definition of an EV3ColorSensor
26          EV3ColorSensor sensor = new EV3ColorSensor(
                SensorPort.S1);
27          sensor.setCurrentMode("RGB");
28          //array to store color-values in
29          float[] color = new float[sensor.sampleSize()];
30
31          while (!Thread.currentThread().isInterrupted()) {
32              //read and store color-values in array
33              sensor.fetchSample(color,0);
34              //display color-values on LCD
35              LCD.drawString("Red component: " + Float.
                    toString(color[0]), 0, 0);
36              LCD.drawString("Green component: " + Float.
                    toString(color[1]), 0, 1);
37              LCD.drawString("Blue component: " + Float.
                    toString(color[2]), 0, 2);
38              //wait for one second
39              try {Thread.sleep(1000);}
40              catch (InterruptedException intExc) {
41                  this.interrupt();
42                  break;
43              }
44          }
45      }
46  }
```

Parallelität

Wenn bisher von parallel ablaufenden Threads die Rede war, handelte es sich genau genommen um eine Scheinparallelität (oder auch virtuelle Parallelität). Aufgrund der Tatsache, dass der EV3 nur einen Prozessor hat, kann zu jedem Zeitpunkt immer nur eine Aufgabe ausgeführt werden. Aufgaben müssen daher grundsätzlich nacheinander ausgeführt werden, um eine quasi-parallele Ausführung von Aufgaben zu ermöglichen. Die Rechenzeit des Prozessors wird auf die ablaufenden Threads aufgeteilt. Für den Nutzer entsteht dadurch der Eindruck, dass alle Aufgaben gleichzeitig erledigt werden. Allerdings ist die Rechenleistung jedes Prozessors begrenzt. Laufen zu viele Threads gleichzeitig ab, kann auch eine quasi-Parallelität nicht mehr gewährleistet werden. Das Abfragen eines Sensors könnte unter Umständen derart verzögert

werden, dass eine sinnvolle Auswertung der Sensorsignale nicht mehr möglich ist.

Scheduler

Um alle Threads gleichmäßig abzuarbeiten, gibt es intelligente Algorithmen, die den Threads die Rechenzeit des Prozessors zuteilen und nach einer gewissen Zeit wieder entziehen. Ein Programm, das einen solchen Algorithmus implementiert, wird Scheduler genannt. Zum Beispiel könnte er den verschiedenen Prozessen zyklisch die Rechenzeit des Prozessors zuteilen[1]. Dieses Verfahren wird »Round Robin« oder »Zeitscheibenverfahren« genannt.

Systemabhängig wird dabei jedem Thread für eine gewisse Zeit der Prozessor zur Verfügung gestellt. Während ein Thread gerade den Prozessor belegt, sind alle anderen angehalten. Wenn ein gestoppter Thread für das Auslesen eines Sensors zuständig ist, werden Veränderungen der Messwerte erst registriert, wenn dieser Thread wieder den Prozessor belegen darf. Treten Veränderungen nur sehr kurz auf, und liegen zufällig genau in dem Zeitrahmen, in dem der Thread unterbrochen ist, so werden diese gar nicht erfasst. In ungünstigen Fällen verursacht dieses Problem also ungenaue oder verzögerte Messergebnisse bei der Nutzung von Sensoren.

Garbage Collector

Es gibt einen weiteren verwaltenden Algorithmus: Den sogenannten Garbage Collector (englisch für: Müllabfuhr). Dieser Algorithmus ist dafür verantwortlich, nicht mehr benötigten Speicher wieder freizugeben. Dadurch wird sichergestellt, dass Programmteile, die nicht mehr benötigt werden, den Speicher nicht sinnlos belasten und die so freigemachten Ressourcen den noch laufenden Threads zur Verfügung gestellt werden können. Dies ist besonders interessant, wenn mit mehreren Threads gearbeitet wird. Sollte ein Thread nicht richtig beendet werden, so belegt er weiter Systemressourcen, obwohl er nicht mehr benötigt wird. Diese Programmreste werden dann automatisch nach einer gewissen Zeit vom Garbage Collector eingesammelt (damit sie keinen Speicherplatz mehr verbrauchen). Der Garbage Collector kann aber auch manuell vom Entwickler aufgerufen werden, um zu bestimmten Zeiten für einen bereinigten Speicher zu sorgen.

1 Die Implementierung eines Schedulers (bzw. zeitlich gesteuerten Ablaufs) wird zum Teil von den Java-Klassen java.util.Timer und java.util.TimerTask übernommen.

Kapitel 6

Erste Schritte in leJOS

6.1 »Hello World«

Das üblicherweise erste Programm beim Lernen einer neuen Programmiersprache ist »Hello World«. Seinen Ursprung hat es eigentlich in der Sprache C, sollte jedoch in keiner Programmiereinführung fehlen – so auch hier nicht. Abbildung 6.1 zeigt die Ausgabe unseres ersten »Hello World«-Programms auf dem EV3-Display. Das Programm 6.1 auf Seite 74 zeigt, wie der String `"Hello World"` auf dem Display ausgegeben wird.

Abbildung 6.1.: LCD des EV3 mit dem »Hallo Welt«-Programm

Neues Projekt anlegen

Nachdem die Entwicklungsumgebung Eclipse wie in Anhang A eingerichtet wurde, kann mit dem Schreiben des ersten Programms begonnen werden. Zunächst wird dazu in Eclipse, wie in Anhang B beschrieben, ein neues Projekt angelegt.

»HelloWorld«-Programm

Um den gewünschten String »HelloWorld« auf dem LCD des EV3 ausgeben zu können, müssen zu Anfang des Programms die zuständigen leJOS-Klassen importiert werden. Die Nachricht »HelloWorld« wird über die LCD-Methode `drawString` auf dem LCD ausgegeben.

Programm 6.1: HalloWelt-Programm

```
1  import lejos.hardware.Button;
2  import lejos.hardware.lcd.LCD;
3
4  /**
5  *This class displays ”Hello World” on the EV3-display
6  */
7  public class HelloWorld {
8      public static void main(String[] args) {
9          /*display string in column 0 and row 0*/
10         LCD.drawString(”HelloWorld”, 0, 0);
11         /*Wait for any key to be pressed*/
12         Button.waitForAnyPress();
13     }
14 }
```

Im Eclipse-Menü wird das aktuell geschriebene Programm über File → Save gespeichert.

6.2 Programm auf dem EV3 ausführen

Verbindung mit dem EV3 herstellen

Um das Programm auf den EV3 übertragen und dort starten zu können muss zunächst eine Verbindung zwischen dem Rechner und dem EV3 bestehen. Dazu können folgende zwei Methoden genutzt werden:

- Netzwerkverbindung zum EV3 über USB herstellen
- Netzwerkverbindung zum EV3 über WLAN herstellen

Beide Methoden werden im Anhang A in Abschnitt A.4 auf Seite 167 und Abschnitt A.5 auf Seite 172 beschrieben und können je nach Vorliebe genutzt werden.

Bei der Nutzung mehrerer EV3s, beispielsweise in Schulklassen, bietet es sich an, die Verbindung des jeweiligen EV3s mit dem PC der Schülerin bzw. des Schülers per USB herzustellen, da dies unkomplizierter ist.

Programm auf dem EV3 starten

Ist der EV3 erfolgreich mit dem PC verbunden, kann das Programm in Eclipse per Rechtsklick im *Package Explorer* auf das Projekt »EV3FirstProject« mittels Run As → LeJOS EV3 Program gestartet werden (Siehe Abbildung 6.2 auf der nächsten Seite). Damit wird das Programm kompiliert, als ausführbare `jar`-Datei gepackt, auf den EV3 kopiert und dort direkt ausgeführt.

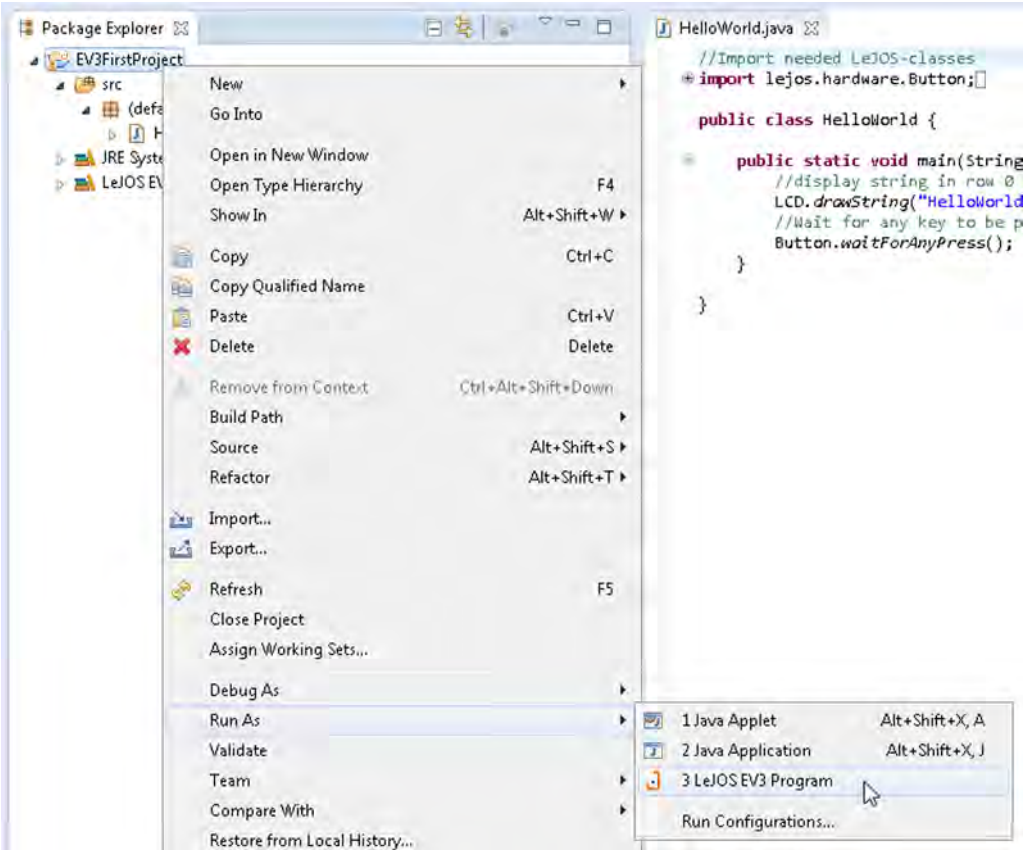


Abbildung 6.2.: Programm auf dem EV3 ausführen

Das Programm befindet sich nun als ausführbare `jar`-Datei[1] auf dem EV3. Möchte man es erneut ausführen kann über den EV3-Menüpunkt `Programs` zu eben dieser `HelloWorld.jar` navigiert werden. Mit ENTER (siehe Abbildung 6.4 auf der nächsten Seite) wird das Programm ausgewählt und mit erneutem ENTER gestartet. Nach einer kurzen Zeit erscheint erneut die Nachricht »Hello World« auf dem LCD-Bildschirm.

1 Dabei handelt es sich um ein sogenanntes »Java Archive«, welches neben dem kompilierten Java-Bytecode auch die benötigten Klassenbibliotheken enthält.

Abbildung 6.3.: EV3-Menü »Programs«

Das Programm kann, seiner Programmierung entsprechend, durch Drücken einer beliebigen Taste beendet werden. Daraufhin kehrt der EV3 zum Hauptmenü zurück.

6.3 leJOS Menü und Tastenbelegung

Nach dem Start zeigt der EV3 das Hauptmenü an, mit `Run-Default` als ausgewähltem Menüpunkt (siehe Abbildung 6.4). Zwischen den verschiedenen Menüpunkten kann mit der RIGHT- und LEFT-Taste des EV3 navigiert werden. Das jeweils mittige Symbol stellt den aktuell angewählten Menüpunkt dar. Um diesen Menüpunkt auszuführen, wird die ENTER-Taste des EV3 gedrückt. Um in das Hauptmenü zurückzugelangen wird die ESCAPE-Taste genutzt. Die Tastenbelegung des EV3 ist ebenfalls in Abbildung 6.4 dargestellt.

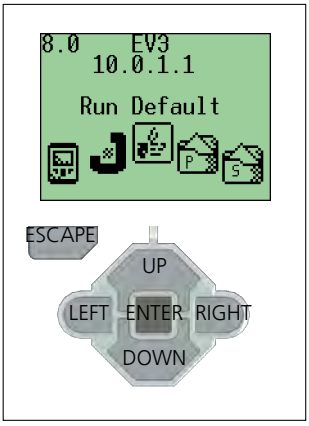


Abbildung 6.4.: Startbildschirm und Tastenbelegung des EV3

Neben der Darstellung der einzelnen Menüpunkte wird auf dem Bildschirm der Name des EV3 angezeigt, der selbst gewählt werden kann (in Abbildung 6.4 auf der vorherigen Seite heißt der EV3 standardmäßig noch »EV3«). Darunter wird die IP-Adresse des eigenen EV3-Geräts angezeigt, welche für USB-Netzwerk- und Bluetooth-Verbindungen genutzt wird, und darunter, falls vorhanden, die IP-Adresse für eine WiFi-Verbindung. Links neben dem Namen des EV3 wird der Ladezustand des Akkus angezeigt.

Run Default

Wird nun beispielsweise der Menüpunkt `Run Default` ausgeführt, wird das als *default* gekennzeichnete Programm gestartet. Wenn kein solches Programm auf dem EV3 existiert, wird für einige Sekunden die Nachricht »`No default set`« angezeigt. Danach gelangt man wieder ins Hauptmenü.

Programs

Ein weiterer Menüpunkt ist *Programs*, rechts neben dem *Run Default*-Symbol. Wird dieser angewählt und mit ENTER bestätigt, werden die auf dem EV3 im Verzeichnis `/home/lejos/programs` verfügbaren Programme sowie andere dort hinterlegte Dateien aufgelistet (siehe Abbildung 6.5).

Durch die Liste kann mit den UP- und DOWN-Tasten navigiert werden. Wird mit ENTER eine der Dateien ausgewählt, bietet der folgende Bildschirm (siehe Abbildung 6.6) einige Optionen für diese Datei, wie `Execute program`, `Delete file` oder `Set as Default`.

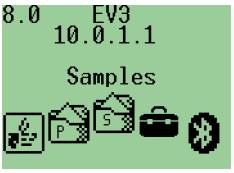
Abbildung 6.5.: Auswahlbildschirm der Dateien

Abbildung 6.6.: Optionen für eine einzelne Datei

Ist die ausgewählte Datei kein Java-Programm mit der Dateiendung `.jar`, stehen weniger Optionen zur Auswahl. Sound-Dateien mit der Endung `.wav` beispielsweise können mit der Option »Play« abgespielt werden anstatt gestartet zu werden. Generell steht aber mindestens die Option »Delete File« zur Verfügung.

Samples-Bildschirm

Ähnlich dem *Files*-Menüpunkt ist es im `Samples`-Menü möglich, Programme auf dem EV3 zu starten. Einziger Unterschied ist, dass die Programme im `Files`-Ordner selbst erstellte Nutzer-Programme sind, wohingegen die im `Samples`-Ordner befindlichen Programme Beispiele aus der leJOS-Community sind.

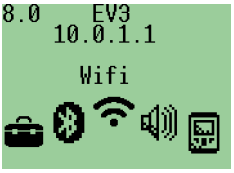


Bluetooth-Bildschirm

Über den Menü-Punkt *Bluetooth* ist es möglich, andere Bluetooth-Geräte zu suchen und diese mit dem EV3 zu verbinden. Der Standard-Pin »1234« des EV3, der bei der Verbindung mit einem anderen Bluetooth-Gerät eingegeben werden muss, kann über das Bluetooth-Menü geändert werden. Auch ist es möglich, die Bluetooth-Verbindung an- oder auszuschalten und sich eine Liste mit schon bekannten Bluetooth-Geräten anzeigen zu lassen.

WiFi-Bildschirm

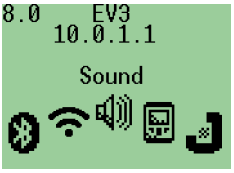
Ist eine WiFi-Verbindung gewünscht, kann diese über den Menüpunkt *WiFi* eingerichtet werden. Nachdem der WiFi-Dongle über den USB-Anschluss mit dem EV3 verbunden ist, muss der EV3 mit einem WiFi-Access-Point verbunden werden.



Eine genaue Beschreibung zur Einrichtung des WLANs zur Verbindung des EV3s mit dem PC folgt in Anhang A.

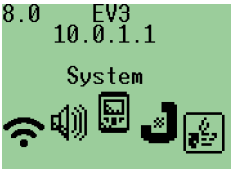
Sounds-Bildschirm

Im Menüpunkt *Sounds* können Einstellungen zur generellen Lautstärke des EV3-Geräts vorgenommen werden, sowie die individuelle Lautstärke, Tonlänge und -frequenz für die einzelnen Tasten des Geräts geändert werden.



System-Bildschirm

Der Menüpunkt *System* erlaubt es dem Nutzer, verschiedene generelle Einstellungen des EV3-Geräts vorzunehmen, wie zum Beispiel:



- Löschen aller im Verzeichnis `/home/lejos/programs` befindlichen Dateien
- An- bzw. Ausschalten des »Auto Run«[2]
- Ändern des Gerätenamens des EV3 (Standardmäßig heißt jeder EV3 zunächst »EV3«)

Als letzter Punkt des Haupmenüs kann der Nutzer über *Version* die aktuell auf dem EV3 installierte leJOS-Version sowie die aktuelle Menü-Version angezeigt bekommen.

Version-Bildschirm

2 Bei aktiviertem Auto Run wird das als default markierte Programm automatisch nach dem hochfahren des EV3 ausgeführt.

Klassen- und Methodenübersicht

Dieses Kapitel gibt eine Übersicht über die für die Programmierung des EV3 wichtigsten Klassen und Methoden. Für jede Klasse wird zunächst eine Übersicht über das repräsentierte Objekt gegeben, danach folgt eine Beschreibung der Klassenmethoden, abschließend wird die Verwendung der Klasse an einem Beispiel verdeutlicht.

Der Abschnitt über die leJOS-Klassen und leJOS-Methoden basiert auf dem von leJOS konzipierten Wiki, welches unter http://sourceforge.net/p/lejos/wiki/Home/ eingesehen werden kann.[1]

7.1 Display

Das Display ist, neben dem Sound und der Hintergrundbeleuchtung der Tasten, die Hauptausgabemöglichkeit des EV3. Auf dem Display können sowohl Textnachrichten als auch Messwerte oder Sensorwerte ausgegeben werden. Dabei handelt es sich um ein schwarz/weiß-LC-Display mit einer Auflösung von 178 x 128 Pixel. Ein Zeichen ist 16px hoch und 10px breit. Somit können in einer Zeile 17 Zeichen untergebracht werden. Durch die Zeichenhöhe ergibt sich, dass maximal acht Zeilen auf dem Display zur Verfügung stehen, beginnend bei Zeile null bis Zeile sieben. Der Ursprung des Koordinatensystems befindet sich, wie in Java üblich, am oberen linken Displayrand (siehe Abbildung 7.1 auf der nächsten Seite).

1 An dieser Stelle sei auch auf die leJOS-API verwiesen, welche unter http://www.lejos.org/ev3/docs/ eingesehen werden kann und eine vollständige Beschreibung aller leJOS-Klassen darstellt.

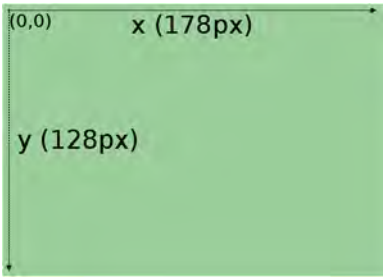


Abbildung 7.1.: LCD-Dimensionen des EV3

Die für das Display verwendete Klasse heisst »LCD« und befindet sich im Package »lejos.hardware.lcd« (zum Einbinden in Eclipse: `import lejos.hardware.lcd.LCD`). Es folgt eine Übersicht über die wichtigsten Methoden zur Verwendung des Displays.

drawString `(String str, int x, int y, boolean inverted)`
gibt einen String auf dem Display in Spalte `x` und Zeile `y` aus. Mit `inverted` kann der Text farblich invertiert (weiße Schrift auf schwarzem Hintergrund) ausgegeben werden. Die Methode ist `static`.

Parameter
- `str` – Der anzuzeigende String
- `x` – X-Koordinate
- `y` – Y-Koordinate
- `inverted` – Optionaler Parameter zur farblichen Invertierung des Strings.

---

drawInt `(int i, int x, int y)`
gibt eine Ganzzahl `i` auf dem Display in Spalte `x` und Zeile `y` aus. Methode ist `static`.

Parameter
- `i` – Die anzuzeigende Ganzzahl
- `x` – X-Koordinate
- `y` – Y-Koordinate

---

drawChar `(char c, int x, int y)`
gibt einen einzelnen Buchstaben auf dem Display in Spalte `x` und Zeile `y` aus. Methode ist `static`.

Parameter
- `c` – Der anzuzeigende Buchstabe
- `x` – X-Koordinate
- `y` – Y-Koordinate

---

clear `()`
löscht die Gesamte Anzeige auf dem Bildschirm. Methode ist `static`.

---

clear `(int y)`
löscht die Anzeige in Zeile `y`. Methode ist `static`.

Parameter
- `y` – Zu löschende Zeile

---

clear `(int x, int y, int n)`
löscht `n` Zeichen, beginnend in Spalte `x` und Zeile `y`. Methode ist `static`.

Parameter
- `x` – X-Koordinate
- `y` – Y-Koordinate
- `n` – Anzahl zu löschender Zeichen

---

scroll `()`
lässt den gesamten Bildschirminhalt um eine Zeile nach oben wandern. Methode ist `static`.

Der Modifier `static` bedeutet, dass es sich hier um sogenannte Klassenmethoden handelt, die verwendet werden können, ohne vorher ein Objekt der Klasse erzeugen zu müssen. Der Aufruf erfolgt an der gewünschten Stelle mittels `LCD.Methodenname(Parameter)`.

Das Programm 7.1 zur Klasse LCD zeigt den Text »Die Loesung aller moeglichen Probleme lautet: 42«. Dabei wird jedes Wort in einer eigenen Zeile dargestellt. In acht Schritten wird der Text zeilenweise nach oben verschoben, wodurch die oberste Zeile immer aus dem Display verschwindet. Zwischen den Schritten wird der *Thread* für 1000ms, also für eine Sekunde, angehalten.

Programm 7.1: LCD-Test

```
1  import lejos.hardware.lcd.LCD;
2  /**
3  * This class displays a text on the EV3-Display.
4  * After being printed, the text is scrolled upwards out
       of the screen.
5  */
6  public class LCDTest {
7      public static void main(String[] args) throws
           Exception {
8          /*Print text line per line*/
9          LCD.drawString("The", 0, 0);
10         LCD.drawString("answer", 0, 1);
11         LCD.drawString("to", 0, 2);
12         LCD.drawString("all", 0, 3);
13         LCD.drawString("questions", 0, 4);
14         LCD.drawString("is: ", 0, 5);
15         /*Print Number 42*/
16         LCD.drawInt(42, 0, 6);
17         /*Scroll seven times*/
18         for (int i = 0; i < 7; i++) {
19             Thread.sleep(1000);
20             LCD.scroll();
21         }
22     }
23 }
```

Die Ausgabe des Programms auf dem Display des EV3 lautet:

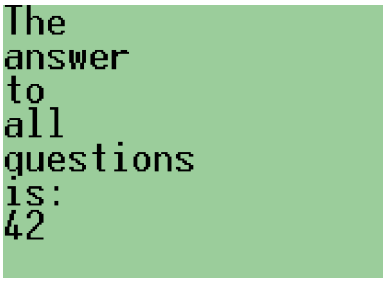


Abbildung 7.2.: Ausgabe des LCD-Test-Programms

7.2 Bedienelemente – EV3 Tasten

Der EV3-Baustein verfügt über sechs Tasten, die direkt unter dem Display platziert sind. Die Tasten sind entsprechend ihrer Ausrichtung benannt, also LEFT, RIGHT, DOWN, UP, ENTER und ESCAPE. Ihre Belegung wurde in Abschnitt 6.3 auf Seite 76 genauer beschrieben. Die Methoden zur Ansteuerung der Tasten befinden sich in der Klasse »Button« aus dem Package »lejos.hardware« (zum Einbinden in Eclipse: `import lejos.hardware.Button`). Sie werden mit `Button.NAME.Methodenname()` angesprochen, wobei der `NAME` entweder LEFT, RIGHT, DOWN, UP, ENTER oder ESCAPE sein kann. Es folgt eine Übersicht über die wichtigsten Methoden zur Programmierung mit den EV3-Tasten:

isDown ()
prüft, ob die Taste zum Zeitpunkt des Methodenaufrufs gedrückt ist.

Rückgabewert
`boolean` – Taste ist gedrückt: `true`; Taste ist nicht gedrückt: `false`

---

isUp ()
prüft, ob die Taste zum Zeitpunkt des Methodenaufrufs oben (**nicht** gedrückt) ist.

Rückgabewert
`boolean` – Taste ist **nicht** gedrückt: `true`; Taste ist gedrückt: `false`

---

waitForPress ()
wartet darauf, dass die mittels NAME angegebene Taste gedrückt wird und fährt unmittelbar mit der Programmausführung fort.

---

waitForPressAndRelease ()
wartet darauf, dass die mittels NAME angegebene Taste gedrückt und wieder losgelassen wird. Es wird erst nach dem Loslassen mit der Programmausführung fortgefahren

---

getID ()
gibt die Ganzzahl-ID der mittels NAME angegebenen Taste zurück.

Rückgabewert

`int` – ID_UP: 1
ID_ENTER: 2
ID_DOWN: 4
ID_RIGHT: 8
ID_LEFT: 16
ID_ESCAPE: 32

---

waitForAnyEvent `()`
wartet darauf, dass eine beliebige Taste gedrückt oder losgelassen wird. Die Methode gibt die ID der Taste zurück. Sie wird nur mittels `Button.Methodenname()` aufgerufen, da sie sich nicht auf eine bestimmte Taste bezieht.

Rückgabewert

`int` – BUTTON_ID

---

waitForAnyPress `()`
wartet darauf, dass eine beliebige Taste gedrückt wird und gibt die ID der Taste zurück, die gedrückt wurde. Die Methode wird ebenfalls nur mittels `Button.Methodenname()` aufgerufen, da sie sich nicht auf eine bestimmte Taste bezieht.

Rückgabewert

`int` – BUTTON_ID

In folgendem Programm 7.2 wird in einer `while`-Schleife auf das Drücken einer Taste gewartet und geprüft welche Taste gedrückt wurde, um dies daraufhin auf dem Display anzuzeigen. Dafür wird die ID der Taste in einer Integer-Variable zwischengespeichert. Beim Drücken der ESCAPE-Taste wird die `while`-Schleife und damit auch das gesamte Programm beendet.

Programm 7.2: Button-Test

```
1 import lejos.hardware.Button;
2 import lejos.hardware.lcd.LCD;
3
4 /**
5 * This class waits for a Button to be pressed and then
      displays its name. Program exits after ESCAPE is
      pressed.
6 */
7 public class ButtonTester {
```

```
 8
 9      public static void main(String[] args) {
10          int buttonId = 0;
11
12          while (buttonId != Button.ID_ESCAPE) {
13              buttonId = Button.waitForAnyPress();
14
15              if (buttonId == Button.ID_LEFT) {
16                  LCD.clear();
17                  LCD.drawString("LEFT pressed", 0, 0);
18              }
19              else if (buttonId == Button.ID_RIGHT) {
20                  LCD.clear();
21                  LCD.drawString("RIGHT pressed", 0, 0);
22              }
23              else if (buttonId == Button.ID_ENTER) {
24                  LCD.clear();
25                  LCD.drawString("ENTER pressed", 0, 0);
26              }
27              else if (buttonId == Button.ID_UP) {
28                  LCD.clear();
29                  LCD.drawString("UP pressed", 0, 0);
30              }
31              else if (buttonId == Button.ID_DOWN) {
32                  LCD.clear();
33                  LCD.drawString("DOWN pressed", 0, 0);
34              }
35          }
36      }
37  }
```

Die Ausgabe des Programms auf dem Display des EV3 lautet:

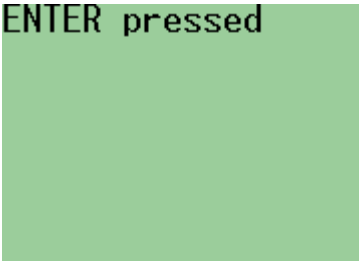


Abbildung 7.3.: Ausgabe des Button-Test-Programms (nach Drücken der ENTER-Taste)

Wie wir in den `if`-Abfragen in Programm 7.2 sehen, kann auf die ID einer Taste direkt in der Klasse »Button« mittels `Button.ID_BUTTONNAME` zugegriffen werden. Sie muss also nicht jedes Mal mittels `Button.BUTTONNAME.getId()` ermittelt werden.

7.3 Lautsprecher

Der EV3 verfügt über einen internen Monolautsprecher, mit dem einzelne Töne und Tonfolgen oder auch WAV-Dateien abzuspielen. Über das Firmware-Menü kann er zudem stumm geschaltet werden. Das beinhaltet Tastentöne und die Wiedergabe von Sounddateien in Programmen. Zur Verwendung des Lautsprechers stellt leJOS die Klasse »Sound« aus dem Package »lejos.hardware« zur Verfügung (zum Einbinden in Eclipse: `import lejos.hardware.Sound`). Diese liefert u.a. folgende `static`-Methoden zur Ansteuerung des Lautsprechers:

beep `()`
spielt einen einzelnen Ton ab.

---

twoBeeps `()`
spielt einen doppelten Ton ab.

---

beepSequence `()`
spielt eine Tonfolge absteigender Höhe ab.

---

beepSequenceUp `()`
spielt eine Tonfolge aufsteigender Höhe ab.

---

buzz `()`
spielt ein tiefes Summen ab.

---

setVolume `(int vol)`
Stellt die Lautstärke auf den angegebenen Wert.

Parameter
`vol` – Lautstärke mit Wertebereich zwischen Null (Stumm) und 100 (max)

---

getVolume `()`
liest analog zu `setVolume` die aktuelle Lautstärke aus.

Rückgabewert
`int` – Aktuelle Lautstärke

---

playTone `(int freq, int duration)`
spielt einen Ton mit einer bestimmten Frequenz für die angegebene Dauer.

Parameter
`freq` – Frequenz in Hertz
`duration` – Dauer des Tons in Millisekunden

---

playSample `(java.io.File file)`
spielt die übergebene Sounddatei ab und gibt die Länge der Soundausgabe zurück. Bei einem Fehler wird ein negativer Wert zurückgegeben.

Parameter
`file` – Abzuspielende WAV-Datei

Rückgabewert
`int` – Länge der Soundausgabe in Millisekunden

Das Programm 7.3 zur Klasse Sound spielt Töne und Tonfolgen verschiedener Frequenzen ab.

Programm 7.3: Sound-Test

```
1  import lejos.hardware.Sound;
2
3  /**
4  * This class makes the EV3 play a series of sounds.
5  */
6  public class SoundTester {
7      public static void main(String[] args) throws
           Exception {
8          int pause = 100;
9
10         Sound.beep();
11         Thread.sleep(pause);
12         Sound.beepSequenceUp();
13         Thread.sleep(pause);
14         for (int i = 1; i < 6; i++) {
15             Sound.playTone(i*440,500);
16         }
17         for (i = 5; i >= 1; i--) {
18             Sound.playTone(i*440,500);
19         }
20         Thread.sleep(pause);
21         Sound.beepSequence();
22         Thread.sleep(pause);
23         Sound.beep();
24     }
25 }
```

7.4 Batterie

Da der EV3 ein mobiles Endgerät ist, wird dieser durch Akkus oder Batterien betrieben. Um einen möglichst fehlerfreien Betrieb zu gewährleisten, sollte vor kritischen Situationen der Ladezustand der Spannungsversorgung geprüft werden. Ferner ist die Batteriestärke wichtig, um die maximale Motorleistung zu erreichen. Informationen über die eingesetzten Akkus oder Batterien bietet die Klasse »Battery« aus dem Package »lejos.hardware« (einzubinden mittels: `import lejos.hardware.battery`). Folgende Methoden können zum Auslesen der Batteriespannung genutzt werden:

getVoltage ()
gibt die aktuelle Batteriespannung in Volt aus. Methode ist `static`.

Rückgabewert
: `float` – Batteriespannung in Volt

---

getVoltageMilliVolt ()
Gibt die aktuelle Batteriespannung in Millivolt aus. Methode ist `static`.

Rückgabewert
: `int` – Batteriespannung in Millivolt

Das Programm 7.4 zur Klasse »Battery« gibt den Batteriestand am Display aus, bis die ENTER-Taste gedrückt wird.

Programm 7.4: Battery-Test

```
1  import lejos.hardware.Battery;
2  import lejos.hardware.Button;
3  import lejos.hardware.lcd.LCD;
4  /**
5  * This class displays the current battery voltage.
6  */
7  public class BatteryTester {
8      public static void main(String[] args) {
9          LCD.drawString(”Battery:”, 0, 0);
10         LCD.drawString(Battery.getVoltage() +” Volt”,0,1)
               ;
11         LCD.drawString(Battery.getVoltageMilliVolt() +”
               mVolt”,0,2);
12         Button.ENTER.waitForPress();
13     }
14 }
```

Die Ausgabe des Programms auf dem Display des EV3 lautet:

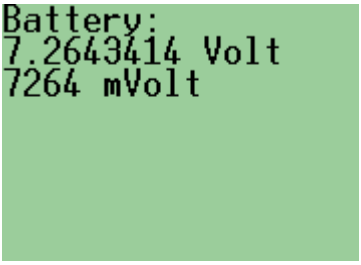


Abbildung 7.4.: Ausgabe des Battery-Test-Programms

7.5 Motoren

Für den EV3 sind zwei Arten von Motoren verfügbar: der *Servomotor* (siehe Abbildung 7.5 rechts) und der *Servomotor medium* (siehe Abbildung 7.5 links).

Abbildung 7.5.: Die Servomotoren des EV3 (LEGO Education 2013)

Großer Servomotor

Der große Servomotor besitzt einen eingebauten Rotationssensor mit einer Genauigkeit von einem Grad. Durch ihn können Messwerte bezüglich des Rotationswinkels erfasst werden. Weiterhin ist es durch einen eingebauten Regelkreis möglich, den Motor mit einer bestimmten Drehgeschwindigkeit laufen zu lassen. So kann die Geschwindigkeit zum Beispiel auf einer Steigung oder unter Last konstant gehalten werden.

Servomotor medium

Der Servomotor medium ist dazu gedacht, in Konstruktionen mit geringerem verfügbaren Platz eingesetzt zu werden sowie bei Anwendungen, die geringere Last und höhere Drehzahlen erfordern. So können kürzere Reaktionszeiten und eine platzsparende Konstruktion erreicht werden. Der Servomotor medium verfügt ebenfalls über einen Rotationssensor und über eine Genauigkeit von einem Grad.

In der nachfolgenden Tabelle 7.1 sind noch einmal die Spezifikationen der beiden Motoren aufgelistet:

	Servomotor	Servomotor medium
Messgenauigkeit	1 Grad	1 Grad
Drehgeschwindigkeit	160 bis 170 U/min	240 bis 250 U/min
Nennmoment	ca. 0,2 Nm (Ø Antriebsmoment)	ca. 0,08 Nm (Ø Antriebsmoment)
Anfahrmoment	ca. 0,4 Nm (maximales, kurzfristig erreichbares Drehmoment)	ca. 0,12 Nm (maximales, kurzfristig erreichbares Drehmoment)
Gewicht	76g	36g
Auto-ID	ja, bei EV3-Software	ja, bei EV3-Software

Tabelle 7.1.: Spezifikationen der beiden EV3-Motoren

Motor-Klasse

In leJOS gibt es zwei Arten von Motoren: die »regulated« und die »unregulated« Motoren.

Die bekannten Motoren des EV3 und auch des NXT gehören zu den »regulated« Motoren. Das heißt, dass die Motoren mittels des Rotationssensors ihre Geschwindigkeit überprüfen und so angewiesen werden können, einen bestimmten Rotationswinkel anzusteuern und eine bestimmte Geschwindigkeit zu fahren. Die Geschwindigkeit hängt bei den Motoren von der Spannung der Batterien ab: Als Faustregel können die Motoren ca. 100 Grad pro Sekunden je Volt erreichen. Mit normalen Alkaline Batterien kann die Geschwindigkeit daher bis zu 900 Grad/Sekunde sein. Bei dem wiederaufladbaren Lithium-Akkupack liegt die Geschwindigkeit bei etwa 740 Grad/Sekunde.

Zu den »unregulated« Motoren gehören beispielsweise die Motoren des alten RCX-Systems und einige Motoren von Zusatzanbietern, wie die Servomotoren von HiTechnic oder Tetrix. Diese besitzen keinen Ro-

tationssensor und können daher nur durch Vorgabe der Power und der Richtungsangabe (vorwärts oder rückwärts) gesteuert werden.

Die Motoren werden über die jeweilige Motor-Klasse und die darin enthaltenen Methoden angesprochen. Die Klasse für den großen Servomotor heißt »EV3LargeRegulatedMotor«, die Klasse für den Servomotor medium heißt »EV3MediumRegulatedMotor«. Beide Klassen erben von der Klasse »BaseRegulatedMotor«, befinden sich im Package »lejos.hardware.motor« und werden wie gewohnt in Eclipse eingebunden. Beim Erzeugen eines Motorobjekts muss dem Konstruktor der Motorport, an den der Motor angeschlossen ist, als Parameter übergeben werden (also konkret: `MotorPort.X`, wobei `X` dann `A`, `B`, `C`, oder `D` sein kann). Die folgende Übersicht beschreibt die wichtigsten Methoden zur Ansteuerung der Servomotoren.

setSpeed `(float speed)`
setzt die Motorgeschwindigkeit in Grad/Sekunde.

Parameter
`speed` – Geschwindigkeit in Grad/Sekunde

---

forward `()`
lässt den Motor vorwärts fahren bis `stop()` aufgerufen wird.

---

backward `()`
lässt den Motor rückwärts fahren bis `stop()` aufgerufen wird.

---

stop `(boolean immediateReturn)`
veranlasst einen sofortigen Stopp und hält den Motor auf Position (lock). Beendet zum Beispiel forward()-Funktionen.

Parameter
`immediateReturn` – Optionaler Parameter; wenn immediateReturn `true` ist, wird sofort mit der Programmausführung fortgefahren, der Motor stoppt von alleine und es wird nicht gewartet bis er vollständig angehalten hat

---

rotate `(int angle, boolean immediateReturn)`
rotiert den Motor um den angegeben Winkel.

Parameter

`angle` – Winkel in Grad

`immediateReturn` – Optionaler Parameter; wenn immediateReturn `true` ist, wird sofort mit der Programmausführung fortgefahren, der Motor stoppt von alleine und es wird nicht gewartet bis er vollständig angehalten hat

---

rotateTo `(int limitAngle, boolean immediateReturn)`
rotiert den Motor auf den angegebenen Winkel relativ zum Nullpunkt des Tachos. Dieser entspricht entweder der Motorposition zum Programmstart oder Position, an der zuletzt `resetTachoCount()` aufgerufen wurde.

Parameter

`limitAngle` – Winkel in Grad

`immediateReturn` – Optionaler Parameter; Wenn immediateReturn `true` ist, wird sofort mit der Programmausführung fortgefahren, der Motor beendet die Rotation im Hintergrund

---

resetTachoCount `()`
setzt den Nullpunkt des Tachos auf die aktuelle Motorposition.

---

getTachoCount `()`
gibt den Tachowert (gedrehte Grad relativ zum Nullpunkt) zurück. Der Tacho verhält sich dabei ähnlich wie der Kilometerzähler eines Autos, mit dem Unterschied, dass Bewegungen in negativer Rotationsrichtung ebenfalls berücksichtigt werden (also vom Tachowert abgezogen werden).

Rückgabewert

`int` – Motorposition in Grad

---

flt `()`
setzt den Motor in den float-Modus. Dadurch stoppt der Motor ohne »lock« und die Motorposition wird nicht länger überwacht. Der Motor ist danach frei drehbar.

---

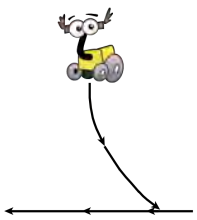
isMoving `()`
prüft, ob der Motor gerade in Bewegung ist.

Rückgabewert

`boolean` – Liefert `true` falls der Motor eine Bewegung ausführt

Analog dazu können die NXT-Servomotoren mittels `NXTRegulated-Motor` angesprochen werden. Die vier Motorports am EV3 werden mit `MotorPort.A`, `MotorPort.B`, `MotorPort.C` und `Motor-Port.D` angesprochen. Die Klasse »MotorPort« befindet sich im Package »lejos.hardware.port« und muss zur Verwendung in Eclipse eingebunden werden.

Bei manchen Aufgaben kann es zweckdienlich sein, einen Motor im »unregulated«-Modus zu nutzen. Dafür kann die Klasse »NXT-Motor« aus dem Package »lejos.hardware.motor« verwendet werden. Die Methoden zu dieser Klasse werden in der leJOS-API unter http://www.lejos.org/ev3/docs/ beschrieben.



Im folgenden Programm 7.5 werden die grundsätzlichen Funktionen der Motoren gezeigt. Wie in der nebenstehenden Abbildung soll der Roboter ein Stück vorwärts fahren, dann eine leichte Linkskurve beschreiben und schließlich ein Stück zurücksetzen. Die Methode `forward()` startet die Motoren, die dann drehen, bis sie explizit von der Methode `stop()` gestoppt werden. Zusätzlich kann die Geschwindigkeit der Motoren eingestellt werden, die dann bis zur nächsten Änderung gültig ist.

Um dieses Beispielprogramm mit einem eigenen Roboter nachzuvollziehen, wird empfohlen das Roberta-Modell (kurz: Roberta) zu verwenden, da dieses auch im weiteren Verlauf des Buches in diversen Beispielen zum Einsatz kommt (siehe Abbildung 7.6 auf der nächsten Seite). Die Roberta kann komplett mit einem LEGO MINDSTORMS Education-Set aufgebaut werden. Die Bauanleitung befindet sich im Roberta-Portal unter http://roberta-home.de/de/node/528 und kann dort angesehen sowie heruntergeladen werden.

Falls kein Education-Set zur Verfügung steht: Die Bauanleitung für den sogenannten »5 Minute Bot« ist unter https://shslab.wikispaces.com/5+Minute+Bot+for+EV3 zu finden. Dieser stellt einen einfachen und schnell aufzubauenden Roboter dar, der ebenfalls von zwei Motoren angetrieben wird.

Abbildung 7.6.: Das Roberta-Modell (kurz: Roberta)

Programm 7.5: Motor-Test

```
1  import lejos.hardware.motor.EV3LargeRegulatedMotor;
2  import lejos.hardware.port.MotorPort;
3  /**
4  * This class makes a Robot with two motors move along a
       certain path.
5  */
6  public class MotorTester {
7      public static void main(String[] args) throws
           Exception {
8          EV3LargeRegulatedMotor motorL = new
               EV3LargeRegulatedMotor(MotorPort.A);
9          EV3LargeRegulatedMotor motorR = new
               EV3LargeRegulatedMotor(MotorPort.B);
10         //set speed to 360 deg/s
11         motorR.setSpeed(360);
12         motorL.setSpeed(360);
13         //forward for 1 seconds
14         motorR.forward();
15         motorL.forward();
16         Thread.sleep(1000);
17         motorR.stop(true);
18         motorL.stop(true);
19         //reduce speed for left motor
20         motorL.setSpeed(250);
21         motorR.forward();
```

```
22          motorL.forward();
23          Thread.sleep(3000);
24          //Forward- movement is finished
25          motorR.stop(true);
26          motorL.stop(true);
27          //set both motors to 360 deg/s again
28          motorR.setSpeed(360);
29          motorL.setSpeed(360);
30          //go backwards for 3 seconds
31          motorR.backward();
32          motorL.backward();
33          Thread.sleep(3000);
34      }
35  }
```

7.6 Sensoren

Die Sensoren des EV3 sind die Schnittstelle, mit der ein Roboter Informationen über seine Umwelt sammeln kann. Im EV3 Spielwaren-Set sind drei Sensoren enthalten: Ein Farbsensor, ein Infrarotsensor mit zugehöriger Infrarot-Fernbedienung und ein Berührungssensor.

Abbildung 7.7.: Die Sensoren des EV3 Spielwaren-Sets (LEGO Education 2013)

Im Education-Set sind dagegen fünf Sensoren enthalten: Der Infrarotsensor mit Fernbedienung wird durch einen Ultraschallsensor ersetzt. Zudem enthält das Education-Set einen Gyrosensor und zwei Berührungssensoren.

Abbildung 7.8.: Die Sensoren des EV3 Education-Sets (LEGO Education 2013)

Im Folgenden wird zunächst darauf eingegangen, wie die Sensoren allgemein in leJOS behandelt werden. Daraufhin folgt zu jedem Sensor ein eigener Unterabschnitt, in dem die spezifischen Sensormethoden erklärt werden und ein Programmbeispiel zur Verwendung des Sensor angeführt wird.

Namensgebung

Um es ProgrammiererInnen einfacher zu machen, die richtige Klasse für den jeweiligen Sensor zu finden, wurden die Klassennamen nach dem Prinzip [Hersteller bzw. Mindstorms-Modell][Sensortyp] vergeben. Die Sensoren für den EV3 heißen also alle »EV3SensorXYZ«. Demnach heißt der Farbsensor für den EV3 »EV3ColorSensor«, der des NXT »NXTColorSensor« und der Farbsensor der Firma HiTechnic wird als »HiTechnicColorSensor« bezeichnet. Falls es mehrere Versionen eines Sensors gibt, wird noch ein dritter Teil an den Namen zur Bezeichnung der Version nach dem Schema [Hersteller bzw. Mindstorms-Modell][Sensortyp][V#] an gehangen. Gäbe es also beispielsweise eine neue Version des Farbsensors für den EV3, würde diese »EV3ColorSensorV2« heißen.

Ports

Die Sensoren werden über die Sensorports mit dem Brick verbunden. Generell kann mit dem Interface `SensorPort` über die Einträge `S1` bis `S4` einer der vier Sensorports angesprochen werden. Der Port, an den der Sensor angeschlossen ist, muss bei der Erstellung eines Sensorobjekts im Konstruktor als Parameter übergeben werden. Konkret also: `SensorPort.SX`, wobei `X` dann `1`, `2`, `3` oder `4` sein kann.

Samples

Die einzelnen Messungen, welche die Sensoren durchführen, werden in leJOS als »Samples« bezeichnet. Ein einzelnes »Sample« besteht aus einem oder mehreren Werten, die im selben Moment aufgenommen wurden. Dazu zählen beispielsweise die gemessene Entfernung eines Ultraschallsensors (ein Messwert) oder die Rot-, Grün-, und Blauanteile einer Messung des Farbsensors (drei Messwerte). Auch wenn der Sensor je Zeitpunkt nur einen Messwert speichert, wird dennoch ein Array genutzt um das »Sample« zu speichern.

Sensormodi

Wie auch nachfolgend in den einzelnen Sektionen zu den Sensoren beschrieben, besitzen einige Sensoren verschiedene Modi, in denen sie betrieben werden können. Der Farbsensor beispielsweise kann im »Farb-ID«-Modus verschiedene Farben erkennen und im Modus »Umgebungslicht« die Lichtintensität messen. leJOS stellt in der aktuellen Version 0.9.0 zwei Möglichkeiten zur Verfügung um die Modi der Sensoren zu nutzen und Messwerte mit ihnen aufzunehmen:

1. Die Verwendung der Methoden der Klasse `BaseSensor`
2. Die Verwendung eines `SampleProvider`-Objekts

BaseSensor

Jedem LEGO-Sensorobjekt stehen die Methoden der Klasse `BaseSensor` durch Vererbung zur Verfügung. Mit ihrer Hilfe kann man einen Sensor in einen gewünschten Modus versetzen und Messdaten erfassen. Diese Vorgehensweise ist sehr intuitiv, da die Samples direkt vom Sensorobjekt kommen und nicht erst den Umweg über einen sogenannten Sample Provider machen (siehe nächsten Unterabschnitt). Aus diesem Grund ist dies die bevorzugte Vorgehensweise, die auch in den Beispielprogrammen dieses Buches verwendet wird, um Messdaten aufzunehmen.

Die Methoden werden direkt vom Sensorobjekt aufgerufen und im Folgenden genauer erklärt:

fetchSample `(float[] sample, int offset)`
speichert die Messdaten des Sensors in das Array `sample`, in das durch den Index `offset` definierte Feld.

Parameter
`sample` – Array zum abspeichern der Messdaten
`offset` – Index, an dem das erste Messdatum im Array abgespeichert werden soll

---

sampleSize `()`
gibt die Anzahl der Elemente je Sample zurück. Diese Zahl ist für jeden Modus unterschiedlich.

Rückgabewert
`int` – Anzahl der Elemente

---

setCurrentMode `(int mode)`
versetzt den Sensor in den der Ganzzahl `mode` entsprechenden Modus. Die zu den Modi gehörenden Ganzzahlen können der Beschreibung der einzelnen Sensoren in diesem Kapitel entnommen werden.

Parameter
`mode` – Einem Modus entsprechende Ganzzahl

---

setCurrentMode `(String modeName)`
versetzt den Sensor in den durch den String `modeName` angegebenen Modus. Die möglichen Modusnamen können der Beschreibung der einzelnen Sensoren in diesem Kapitel entnommen werden.

Parameter
`modeName` – Name eines Modus als String

Sample-Provider

Eine weitere Möglichkeit um Messwerte zu erfassen verwendet die zuvor schon erwähnten `SampleProvider`. Möchte man einen Sensor in einem bestimmten Modus nutzen, muss mit Hilfe des Sensorobjekts ein Objekt vom Typ `SampleProvider` erzeugt werden, welches den Modus repräsentiert. Dieser `SampleProvider` kann dann mittels `fetchSample()` Messdaten erfassen. So kann ein Sensor auch verschiedene Modi innerhalb eines Programms benutzen, ohne dass er manuell dafür konfiguriert werden muss. Zur Erfassung von Messdaten mit Hilfe eines `SampleProviders` stehen also folgende Methoden zur Verfügung:

fetchSample `(float[] sample, int offset)`
speichert die Messdaten des Sensors in das Array `sample`, in das durch den Index `offset` definierte Feld.

Parameter
`sample` – Array zum abspeichern der Messdaten.
`offset` – Index, an dem das erste Messdatum im Array abgespeichert werden soll.

---

sampleSize `()`
gibt die Anzahl der Elemente je Sample zurück. Diese Anzahl ist für jeden Modus konstant.

Rückgabewert
`int` – Anzahl der Elemente

Bei manchen Sensoren benötigt der Wechsel zwischen zwei Modi eine gewisse Zeit zur internen (Um-)Konfiguration des Sensors. Dies sollte beim Programmieren unbedingt beachtet werden, wenn häufig zwischen den Modi gewechselt wird.

Farbsensor

Der Farbsensor des EV3 ist ein digitaler Sensor, der vier verschiedene Modi unterstützt. Zum einen kann der Sensor im »Farb-ID«-Modus Farben erkennen, zum anderen in den beiden Modi »Rotlicht« und »Umgebungslicht« die Lichtintensität messen. Darüberhinaus wird im »RGB«-Modus die Intensität der drei Grundfarben Rot, Grün und Blau gemessen. Der Sensor heisst in leJOS »EV3ColorSensor« und befindet sich im Package »lejos.hardware.sensor«.

Der Farbsensor erkennt im »Farb-ID«-Modus die sieben Farben Rot, Gelb, Grün, Blau, Braun, Schwarz und Weiß sowie »keine Farbe«.

Für den Modus »Rotlicht« ist der Farbsensor mit einem Rotlicht ausgestattet, welches die zu messende Fläche anstrahlt. Das dadurch reflektierte Licht beurteilt der Sensor dann auf einer Skala von 0 (sehr dunkel) bis 1 (sehr hell).

Dieselbe Skala wird für den Modus »Umgebungslicht« genutzt, bei dem das ins Sensorfenster eindringende Umgebungslicht gemessen wird.

Im »RGB«-Modus strahlt der Sensor rotes, grünes und blaues Licht ab und misst die jeweiligen Farbanteile des reflektierten Lichts wiederum auf eben genannter Skala. Um die höchstmögliche Genauigkeit bei der Messung zu erzielen, sollte der Sensor im rechten Winkel zu dem zu messenden Objekt und möglichst nah an dessen Oberfläche positioniert werden.

Folgende Tabelle 7.2 gibt Übersicht über die Modi des Farbsensors. Die angegebenen Parameter `mode` und `modeName` können mit den Methoden der Klasse `BaseSensor` verwendet werden um den Sensor in den gewünschten Modus zu versetzen.

Modus	`int mode`	`String modeName`
»Farb-ID«	0	Color ID
»Rotlicht«	1	Red
»RGB«	2	RGB
»Umgebungslicht«	3	Ambient

Tabelle 7.2.: Modi des Farbsensors

leJOS stellt zur Verwendung des Farbsensors folgende Methoden zur Verfügung:

getColorID ()
gibt einen Integer-Wert zurück, der einer Farbe entspricht. Diese Methode kann unabhänging vom momentan verwendeten Modus des Sensors verwendet werden.

Rückgabewert

`int` –	-1	:	Color.NONE
	0	:	Color.RED
	1	:	Color.GREEN
	2	:	Color.BLUE
	3	:	Color.YELLOW
	6	:	Color.WHITE
	7	:	Color.BLACK
	13	:	Color.BROWN

---

getAmbientMode ()
erstellt ein `SampleProvider`-Objekt für den »Umgebungslicht«-Modus und liefert dieses zurück.

Rückgabewert
`SensorMode` – Sample Provider für den »Umgebungslicht«-Modus

Sample
`sample[0]` – Stärke des auf den Sensor fallenden Lichts mit Fließkommawerten zwischen 0 und 1

---

getColorIDMode ()
erstellt ein `SampleProvider`-Objekt für den »Farb-ID«-Modus und liefert dieses zurück.

Rückgabewert
`SensorMode` – Sample Provider für den »Farb-ID«-Modus

Sample
`sample[0]` – Fließkommazahl mit Ganzzahlwerten zwischen -1 und 13, die einer Farbe entsprechen (Siehe Tabelle unter **getColorID** ())

---

getRedMode ()
erstellt ein `SampleProvider`-Objekt für den »Rotlicht«-Modus und

liefert dieses zurück. Dabei wird Objekt mit rotem Licht bestrahlt und die Stärke des reflektierten Lichts gemessen.

Rückgabewert

`SensorMode` – Sample Provider für den »Rotlicht«-Modus

Sample

`sample[0]` – Stärke des reflektierten Lichts mit Fließkommawerten zwischen 0 und 1

---

getRGBMode `()`
erstellt ein `SampleProvider`-Objekt für den »RGB«-Modus, der RGB-Werte zurückgibt und liefert dieses zurück. Dabei wird das Objekt mit rotem, grünem und blauem Licht bestrahlt und die Stärke des reflektierten Lichts gemessen.

Rückgabewert

`SensorMode` – Sample Provider für den »RGB«-Modus

Sample

`sample[0]` – Rotanteil als Fließkommazahl, mit Werten zwischen 0 und 1
`sample[1]` – Grünanteil als Fließkommazahl, mit Werten zwischen 0 und 1
`sample[2]` – Blauanteil als Fließkommazahl, mit Werten zwischen 0 und 1

In folgendem Programm 7.6 wird eine Farbsonde, bestehend aus einem EV3 und dem Farbsensor am Sensorport S1, programmiert. Beim Drücken der ENTER-Taste sollen die gemessenen Farbwerte des Sensors für Rot, Grün und Blau auf dem Display ausgegeben werden. Der Farbsensor wird also im RGB-Modus betrieben. Beim Drücken einer beliebigen anderen Taste wird das Programm beendet. Dafür wird die Klasse »ColorProbe« erstellt. Sie enthält eine Variable für den Farbsensor und eine weitere für das Sample. Der Sensor wird im Konstruktor in den »RGB«-Modus versetzt. Des weiteren enthält die Klasse eine Methode, die für das Auslesen der Farbwerte und die Ausgabe auf dem Display zuständig ist. Diese wird aus der `main`-Methode aufgerufen.

Programm 7.6: Beispielprogramm zum EV3ColorSensor

```
1 import lejos.hardware.Button;
2 import lejos.hardware.lcd.LCD;
3 import lejos.hardware.port.SensorPort;
4 import lejos.hardware.sensor.EV3ColorSensor;
5 import lejos.robotics.SampleProvider;
```

```
6  /**
7  * This class describes a color probe.
8  * It fetches a RGB-sample after ENTER is pressed.
9  */
10 public class ColorProbe {
11
12     private EV3ColorSensor colorSensor = new
           EV3ColorSensor(SensorPort.S1);
13     private float[] sample
14     /**
15     * Constructor for setting sensor-mode to "RGB"
16     */
17     ColorProbe() {
18         colorSensor.setCurrentMode("RGB");
19         sample = new float[colorSensor.sampleSize()];
20     }
21     /**
22     * Method for getting a sample and printing the RGB-
           values
23     */
24     private void getAndPrintData() {
25         LCD.drawString("ENTER for sample", 0, 0);
26         while (Button.waitForAnyPress() == Button.
               ID_ENTER) {
27             colorSensor.fetchSample(sample, 0);
28             LCD.drawString("Red: " + sample[0], 0, 1);
29             LCD.drawString("Green: " + sample[1], 0, 2);
30             LCD.drawString("Blue: " + sample[2], 0, 3);
31         }
32     }
33
34     public static void main(String[] args) {
35         ColorProbe ourProbe = new ColorProbe();
36         ourProbe.getAndPrintData();
37     }
38 }
```

Berührungssensor

Der Berührungssensor ist der einfachste der vier mitgelieferten Sensoren. Während alle drei anderen Sensoren einen quantitativen Wert zurückgeben, kann der Tastsensor nur erfassen, ob er gedrückt ist oder nicht. Die Klasse, die den Berührungssensor beschreibt heisst »EV3TouchSensor« und befindet sich ebenfalls im Package »lejos.hardware.sensor«. Da es nur einen Modus (»Touch«) gibt, muss der Sensor theoretisch nach Instanziierung nicht erst in diesen Modus versetzt werden um Samples aufzunehmen. Es wird allerdings empfohlen immer einen Modus explizit anzugeben, da es sonst bei der Initialisierung des Sensors zu Fehlern kommen kann.

Falls gewünscht erhält man den entsprechenden `SampleProvider` mittels:

getTouchMode `()`
liefert ein `SampleProvider`-Objekt für den »Touch«-Modus.

Rückgabewert

`SensorMode` – Sample Provider für den »Touch«-Modus

Sample

`sample[0]` – Fließkommazahl, die den Zustand des Touchsensor zum Zeitpunkt der Messung darstellt: 1.0 falls der Sensor gedrückt war, 0.0 wenn nicht

In folgendem Programm 7.7 wird die Farbsonde aus Programm 7.6 um einen Touchsensor am Sensorport S2 erweitert. Es soll nun jedes Mal, wenn der Touchsensor gedrückt wird, eine Farbmessung durchgeführt und die Farbanteile auf dem Display ausgegeben werden. Damit das Programm durch Drücken der ENTER-Taste beendet wird, sind entsprechende Abfragen in der `while`- und `do`- Schleife eingefügt worden.

Programm 7.7: Beispielprogramm zum EV3TouchSensor

```
1  import lejos.hardware.Button;
2  import lejos.hardware.lcd.LCD;
3  import lejos.hardware.port.SensorPort;
4  import lejos.hardware.sensor.EV3ColorSensor;
5  import lejos.hardware.sensor.EV3TouchSensor;
6
7  /**
8  * This class implements a color probe, that is triggered
       by a EV3-TouchSensor
9  */
10 public class ColorProbeTouch {
11
12     EV3ColorSensor colorSensor = new EV3ColorSensor(
           SensorPort.S1);
13     EV3TouchSensor touchSensor = new EV3TouchSensor(
           SensorPort.S2);
14     private float[] colorSample;
15     private float[] touchSample;
16     /**
17     * Constructor for getting RGBMode- Sample Provider
18     */
19     ColorProbeTouch() {
20         colorSensor.setCurrentMode("RGB");
21         touchSensor.setCurrentMode("Touch");
22         colorSample = new float[colorSensor.sampleSize()
               ];
23         touchSample = new float[touchSensor.sampleSize()
               ];
```

```
24      }
25      /**
26      * Method for getting a sample and printing the color-
            values
27      */
28      public void getAndPrintData() {
29          while (Button.ENTER.isUp()) {
30              LCD.drawString("Touch for sample",0,0);
31              do {
32                  touchSensor.fetchSample(touchSample,0);
33              } while ((touchSample[0] == 0.0) && Button.
                    ENTER.isUp());
34              touchSample[0] = 0.0F;
35              colorSensor.fetchSample(colorSample, 0);
36              LCD.drawString("Red: " + colorSample[0], 0,
                    1);
37              LCD.drawString("Green: " + colorSample[1], 0,
                     2);
38              LCD.drawString("Blue: " + colorSample[2], 0,
                    3);
39          }
40      }
41
42      public static void main(String[] args) {
43          ColorProbeTouch ourProbe = new ColorProbeTouch();
44          ourProbe.getAndPrintData();
45      }
46  }
```

Ultraschall-sensor

Der Ultraschallsensor kann die Distanz zu einem Objekt erkennen. Er misst die Distanz eines Objektes von 3cm bis 250cm mit einer Genauigkeit von +/- 1 cm. Bei leJOS wird allerdings die Einheit Meter verwendet, ein Sample liegt also auf einer Skala zwischen 0,03 und 2,50.

Um die Entfernung zu messen, dient der »Distanz«-Modus, dessen Aktivität durch ein dauerhaft leuchtendes rotes Licht signalisiert wird. In diesem Modus sendet der Ultraschallsensor hochfrequente Schallwellen aus. Die Zeit, welche die Schallwellen benötigen, um von einem Objekt reflektiert und wieder zum Sensor zurückzukommen, wird gemessen und in eine Distanz umgerechnet.

Da die Messung nur bei einer direkten Reflexion der Schallwellen zuverlässig möglich ist, kann es vorkommen, dass strukturierte oder abgerundete Oberflächen und zu kleine Objekte nicht oder nur schwer erkannt werden.

Der Sensor unterstützt neben dem gerade beschriebenen Modus »Distanz« noch den »Passiv«-Modus. In diesem Modus kann er andere aktive Ultraschallsensoren erkennen, die sich in der Nähe befinden. Damit keine eigenen Schallwellen fälschlicherweise für einen anderen aktiven Sensor gehalten werden, sendet der Ultraschallsensor im »Passiv«-Modus keine eigenen Schallwellen aus. Erkennbar ist dieser Modus am blinkenden roten Licht.

Modus	`int mode`	`String modeName`
»Distanz«	0	Distance
»Passiv«	1	Listen

Tabelle 7.3.: Modi des Ultraschallsensors

Zur Verwendung des Ultraschallsensors stellt leJOS die im Folgenden beschriebenen Methoden zur Verfügung:

disable `()`
schaltet den Sensor sowie die Status-LED aus.

---

enable `()`
schaltet den Sensor sowie die Status-LED nach einem Aufruf von `disable()` wieder ein.

---

getDistanceMode `()`
erstellt ein `SampleProvider`-Objekt für den »Distanz«-Modus und liefert dieses zurück.

Rückgabewert
: `SampleProvider` – Sample Provider für den »Distanz«-Modus

Sample
: `sample[0]` – Entfernung als Fließkommazahl in Metern mit Werten von 0,03 bis 2,5

---

getListenMode `()`
erstellt ein `SampleProvider`-Objekt für den »Passiv«-Modus und liefert dieses zurück.

Rückgabewert
: `SampleProvider` – Sample Provider für den »Passiv«-Modus

Sample

`sample[0]` – Fließkommazahl: 1.0 falls eine Ultraschallquelle detektiert wird, 0.0 sonst

---

isEnabled `()`

gibt zurück, ob der Sensor angeschaltet ist oder nicht.

Rückgabewert

`boolean` – Liefert `true` bei eingeschaltetem Sensor, `false` bei ausgeschaltetem Sensor

Zur Veranschaulichung der Verwendung des EV3-Ultraschallsensors wird in folgendem Programm 7.8 eine Roberta darauf programmiert, in Abständen von 250ms eine Entfernungsmessung durchzuführen. Die resultierenden Entfernungen werden in Frequenzen umgerechnet, die dann ebenfalls für 250ms als Ton abgespielt werden. Durch Veränderung der Entfernung zu einem Objekt, zum Beispiel der Hand oder einer Wand, kann der Programmierer – jetzt als Dirigent – dem Roboter Anweisungen zum »musizieren« geben.

Programm 7.8: Beispielprogramm zum EV3UltrasonicSensor

```
1  import lejos.hardware.Button;
2  import lejos.hardware.Sound;
3  import lejos.hardware.lcd.LCD;
4  import lejos.hardware.port.SensorPort;
5  import lejos.hardware.sensor.EV3UltrasonicSensor;
6  import lejos.utility.Delay;
7
8  public class UltrasonicRoberta {
9
10     private EV3UltrasonicSensor uSensor = new
           EV3UltrasonicSensor(SensorPort.S4);
11     private float[] sample;
12     /**
13     * Constructor for setting sensor-mode to "Distance"
14     */
15     UltrasonicRoberta() {
16         uSensor.setCurrentMode("Distance");
17         sample = new float[uSensor.sampleSize()]
18     }
19     /**
20     * This method plays 250ms long tones with different
           frequencies.
21     * The frequencies depend on measured distances from
           the ultrasonic sensor in distance-mode.
22     */
23     private void sing() {
24         int frequency;
```

```
25
26          while (Button.ENTER.isUp()) {
27              uSensor.fetchSample(sample, 0);
28              frequency = (int) (sample[0] * 2000);
29              LCD.drawString(""+frequency, 0, 0);
30              Sound.playTone(frequency, 250);
31              Delay.msDelay(250);
32              LCD.clear();
33          }
34      }
35
36      public static void main(String[] args) {
37          UltrasonicRoberta roberta = new UltrasonicRoberta
                ();
38          roberta.sing();
39      }
40  }
```

Infrarotsensor

Der digitale Infrarotsensor ist in der Lage, Infrarotlicht, das von Festkörperobjekten reflektiert wird, zu erkennen und so grob die Entfernung zu diesen abzuschätzen. Ausserdem kann er die Infrarotsignale der Fernbedienung wahrnehmen. leJOS stellt die Klasse »EV3IRSensor« aus dem Package »lejos.hardware.sensor« zur Verwendung des Infrarotsensors zur Verfügung.

Der Infrarotsensor kann in zwei verschiedenen Modi betrieben werden. Der sogenannte »Distanz«-Modus dient der Entfernungsmessung zwischen dem Sensor und einem Objekt. Die von dem Objekt reflektierten Lichtwellen werden gemessen und ein Distanz-Wert auf einer Skala von 0 (sehr nah) bis 100 (sehr entfernt) zurückgegeben (es wird also keine genaue Maßeinheit verwendet). Die maximale Entfernung, die der Sensor erkennen kann, liegt bei circa 70cm zwischen Sensor und Objekt. Dies kann je nach Größe, Form und Oberfläche des Objekts allerdings stark abweichen.

Der »Suchen«-Modus dient der Ortung eines Signals der Infrarot-Fernbedienung des EV3. Die maximale Distanz, die zwischen Sensor und Fernbedienung liegen darf, damit der Sensor noch Signale erkennen kann, liegt bei circa 200 cm (in Blickrichtung des Sensors). Der Sensor kann die ungefähre Richtung und Distanz eines empfangenen Signals schätzen und teilt dieses in die zwei Skalen von 0 (sehr nah) und 100 (sehr entfernt) für die Distanz und von -25 (linke Seite) bis 25 (rechte Seite) für die Richtung ein. Ein aufgenommenes Sample enthält somit

acht Werte, zwei für jeden der vier Kanäle: Einen für die Entfernung und einen für die Richtung aus der das Signal kommt.

Eine 0 bei der Richtung bedeutet also, dass sich die Quelle des Signals frontal vor dem Sensor befindet.

Modus	`int mode`	`String modeName`
»Distanz«	0	Distance
»Suchen«	1	Seek

Tabelle 7.4.: Modi des Infrarotsensor

Die Fernbedienung kann Infrarotsignale auf vier verschiedenen Kanälen (engl. channel) senden, damit keine Störungen beim gleichzeitigen Fernsteuern mehrerer Roboter auftreten. Welcher Kanal benutzt werden soll, kann über den roten Schieberegler eingestellt werden. Wird die große, graue Taste der Fernbedienung betätigt, schaltet sie in den sogenannten Beacon- oder Signal-Modus (angezeigt durch dauerhaftes Leuchten der Status-LED). Die Fernbedienung sendet dann kontinuierlich Infrarot-Signale, die vom Sensor im »Suchen«-Modus geortet werden können.

Um beispielsweise einen Roboter fernzusteuern, können mit der Fernbedienung verschiedene Signale (die dem Druck einer Taste oder der Kombination von Tasten entsprechen) an den EV3 gesendet werden. Mit Hilfe der im Folgenden beschriebenen Methode `getRemoteCommand`, können diese Befehle erkannt und dann auf beliebige Weise darauf reagiert werden. Die vom Sensor erkennbaren Kombinationen sind in Tabelle 7.5 auf der nächsten Seite aufgeführt:

Befehl als Integer	Gedrückte Taste(n)
0	Keine Taste (Signal-Modus ist deaktiviert)
1	Taste 1 (oben-links)
2	Taste 2 (unten-links)
3	Taste 3 (oben-rechts)
4	Taste 4 (unten-rechts)
5	Taste 1 **und** Taste 3
6	Taste 1 **und** Taste 4
7	Taste 2 **und** Taste 3
8	Taste 2 **und** Taste 4
9	Große Beacon-Taste (Signal-Modus ist aktiviert)
10	Taste 1 **und** Taste 2
11	Taste 3 **und** Taste 4

Tabelle 7.5.: Kommandos der IR-Fernbedienung

Es folgt eine Übersicht der wichtigsten Methoden aus leJOS zur Verwendung des EV3-Infrarotsensors.

getRemoteCommand `(int chan)`
gibt den aktuell über den Kanal `chan` empfangenen Befehl als eine der oben beschriebenen Integer-Zahlen zurück.

Parameter
`chan` – Übertragungskanal (0, 1, 2, oder 3)

Rückgabewert
`int` – Taste bzw. Tastenkombination als Ganzzahl codiert. Siehe Tabelle 7.5

---

getDistanceMode `()`
erstellt ein `SampleProvider`-Objekt für den »Distanz«-Modus und liefert dieses zurück.

Rückgabewert
`SensorMode` – Sample Provider für den »Distanz«-Modus

Sample
`sample[0]` – Entfernung als Fließkommazahl ganzzahligen Werten von 1.0 bis 100.0 (keine genaue Maßeinheit vorhanden)

---

getSeekMode ()
Erstellt ein `SampleProvider`-Objekt für den »Suchen«-Modus und liefert dieses zurück. Rückgabewert ist die ungefähre Richtung und Distanz zu einem oder mehreren Infrarotsendern.

Rückgabewert

`SensorMode` – Sample Provider für den Signal-Modus

Sample

`sample[0]` – Entfernung des auf Kanal 1 empfangenen Signals mit werten zwischen 0.0 und 100.0
`sample[1]` – Richtung des auf Kanal 1 empfangenen Signals mit werten zwischen -25.0 (ganz links) und 25.0 (ganz rechts)
`sample[2]` – Entfernung für Kanal 2
`sample[3]` – Richtung für Kanal 2
`sample[4]` – Entfernung für Kanal 3
`sample[5]` – Richtung für Kanal 3
`sample[6]` – Entfernung für Kanal 4
`sample[7]` – Richtung für Kanal 4

Ein Beispiel zum Infrarotsensor kann im Roberta-Portal zu diesem Band unter www.roberta-home.de/javaband-ev3 gefunden werden.

Gyrosensor

Der Gyrosensor, von LEGO auch Kreiselsensor genannt, ist ein weiterer digitaler Sensor und erkennt Drehbewegungen um seine eigene **vertikale** Achse mit einer Frequenz von 1 kHz. Dabei kann entweder die Drehrate in Grad pro Sekunde (maximal 440 Grad/Sekunde), oder der Gesamtdrehwinkel in Grad mit einer Genauigkeit von +/- 3 Grad gemessen werden. leJOS stellt zur Verwendung des Gyrosensors die Klasse »EV3GyroSensor« aus dem Package »lejos.hardware.sensor« zur Verfügung.

Modus	`int mode`	`String modeName`
»Winkel«	0	Angle
»Winkelgeschwindigkeit«	1	Rate
»Winkel und Winkelgeschwindigkeit«	2	Rate and Angle

Tabelle 7.6.: Modi des Gyrosensors

Es folgt eine Beschreibung der wichtigsten Methoden dieser Klasse:

getAngleMode ()
liefert einen Sample Provider für den »Winkel«-Modus. In diesem Modus misst der Sensor den Winkel in Bezug auf seine Ausgangsposition. Ein positiver Winkel bedeutet dabei eine Orientierung nach links und ein negativer Winkel eine Orientierung nach rechts (gemessen in Grad).

Rückgabewert
`SampleProvider` – Sample Provider für den Winkel-Modus

Sample
`sample[0]` – Winkel in Bezug auf die Ausgangsposition des Sensors

---

getRateMode ()
liefert einen Sample Provider für den Modus »Winkelgeschwindigkeit«. Hier wird die Geschwindigkeit der Drehung in Grad pro Sekunde gemessen. Ein positiver Wert bedeutet eine Drehung im Uhrzeigersinn, ein negativer Wert gegen den Uhrzeigersinn.

Rückgabewert
`SampleProvider` – Sample Provider für den Modus »Winkelgeschwindigkeit«

Sample
`sample[0]` – Winkelgeschwindigkeit in Grad/Sekunde

---

getAngleAndRateMode ()
liefert einen Sample Provider für den Modus »Winkel und Winkelgeschwindigkeit«. Hier wird die Geschwindigkeit der Drehung und der aktuelle Winkel in Bezug auf die Ausgangsposition gemessen.

Rückgabewert

`SampleProvider` – Sample Provider für den Modus »Winkel und Winkelgeschwindigkeit«

Sample

`sample[0]` – Winkelgeschwindigkeit in Grad/Sekunde
`sample[1]` – Winkel in Bezug auf die Ausgangsposition des Sensors

In folgendem Programm 7.9 wird zur Veranschaulichung der Benutzung des Gyrosensors wieder das Roberta-Robotermodell verwendet. Diese wird um einen Gyrosensor am Sensorport S3 erweitert. Der Sensor muss so angebracht werden, dass er parallel zur Oberfläche, auf der sich der Roboter bewegen soll, ausgerichtet ist. Die Roberta soll ein Quadrat abfahren indem sie sich für zwei Sekunden geradeaus bewegt, sich anschließend mit Hilfe des Gyrosensors um 90 Grad dreht und diesen Vorgang noch drei Mal wiederholt. Der Gyrosensor ist hier insofern praktisch, da es sonst schwierig ist mit den bisher kennengelernten Methoden eine bestimmte Drehung mit einem zweimotorigen (differential angetriebenen) Roboter durchzuführen. Auf Grund von Messungenauigkeiten wird das Quadrat auch mit dem Gyrosensor nicht ganz genau abgefahren. Ein wenig Feintuning ist also am Winkel (`ANGLE`) nötig. Besser ist eine solche Aufgabe mit der im Kapitel 8 ab Seite 117 vorgestellten »DifferentialPilot«-Klasse zu lösen.

Der Sensor wird in der aktuellen leJOS-Version 0.9.0 bei der ersten Benutzung der `fetchSample`-Methode initialisiert. Dabei muss er absolut ruhig gehalten werden, damit er die richtige Ausgangsposition ermitteln kann. Aus diesem Grund wird die Methode schon im Konstruktor des Beispiels einmal aufgerufen und danach eine Sekunde gewartet. Auch wenn ein Programm den Sensor-Modus wechselt, führt der Sensor intern einen Reset durch, bei dem dieser ebenfalls regungslos sein muss, um anschließend keine falschen Werte zu liefern.

Programm 7.9: Beispielprogramm zum EV3GyroSensor

```
1  import lejos.hardware.motor.EV3LargeRegulatedMotor;
2  import lejos.hardware.port.MotorPort;
3  import lejos.hardware.port.SensorPort;
4  import lejos.hardware.sensor.EV3GyroSensor;
5  import lejos.utility.Delay;
6  /**
7   * This class describes a Roberta-robot, that uses a
        gyrosensor to move along a squared path.
8   */
9  public class GyRoberta {
10
11      private EV3LargeRegulatedMotor engineR = new
            EV3LargeRegulatedMotor(MotorPort.B);
12      private EV3LargeRegulatedMotor engineL = new
            EV3LargeRegulatedMotor(MotorPort.C);
13      private EV3GyroSensor gyroSensor = new EV3GyroSensor(
            SensorPort.S3);
14
15      private float[] sample;
16
17      GyRoberta() {
18          gyroSensor.setCurrentMode("Angle");
19          sample = new float[gyroSensor.sampleSize()];
20          gyroSensor.fetchSample(sample, 0);
21          Delay.msDelay(1000);
22      }
23
24      private void driveSquare() {
25          engineR.setSpeed(250);
26          engineL.setSpeed(250);
27          final int ANGLE = 90;
28
29          for (int i = 1; i <= 4; i++) {
30              engineR.forward();
31              engineL.forward();
32              Delay.msDelay(2000);
33              engineR.stop(true);
34               do {
35                  gyroSensor.fetchSample(sample, 0);
36              } while (sample[0] > (-ANGLE * i));
37              engineL.stop(true);
38          }
39      }
40
41      public static void main(String[] args) {
42          GyRoberta roberta = new GyRoberta();
43          roberta.driveSquare();
44      }
45 }
```

Kapitel 8

leJOS-Robotikklassen

Neben grundlegenden Funktionen, wie dem Auslesen von Sensordaten oder der Ansteuerung von Motoren, bietet leJOS die Möglichkeit erweiterte Konzepte aus der Robotik umzusetzen. Dazu werden im Folgenden ausgewählte Klassen aus dem Packet »lejos.robotics« näher behandelt:

- Die Klasse `DifferentialPilot`
- Fünf Filter-Klassen
- Die Klassen der Subsumption-Architektur

Neben dem »DifferentialPilot« zur Ansteuerung von zwei zusammenarbeitenden Motoren wird vor allem die »Subsumption«-Architektur beschrieben, mit deren Hilfe Roboter mit komplexeren Verhaltensmustern programmiert werden können. Darüber hinaus wird noch die Klasse `Stopwatch` vorgestellt, die für Zeitmessungen zuständig ist und aus dem Packet »lejos.utility« stammt.

8.1 DifferentialPilot

Viele der mobilen EV3-Roboter verwenden zur Fortbewegung zwei Räder, die direkt durch je einen großen EV3-Servomotor angetrieben werden (z.B. Roberta). Mit den bisher verfügbaren Methoden können wir zwar jede mögliche Bewegung des Roboters realisieren, allerdings kann dies sehr schnell umständlich werden, da jeder Motor einzeln angesprochen werden muss. Insbesondere wenn sich der Roboter auf Kreisbögen bewegen soll, muss für jeden Motor eine eigene Rotationsgeschwindigkeit berechnet werden. Aus diesem Grund stellt leJOS die Klasse »DifferentialPilot« aus dem Paket »lejos.robotics.navigation« zur Verfügung. Sie dient als Schnittstelle, über die dem Roboter eine gewünschte Bewegung vorgegeben werden kann. Die Ansteuerung der einzelnen Motoren erfolgt dabei automatisch.

Die notwendige Rotation der einzelnen Motoren zur Ausführung einer gewünschten Bewegung sind vom Durchmesser der verwendeten Räder und von deren Abstand abhängig. Diese Werte müssen für je-

den Roboter ausgemessen und beim Erstellen eines »DifferentialPilot«-Objekts mit Hilfe folgender Konstruktoren übergeben werden.

Konstruktoren

DifferentialPilot`(double wheelDiameter, double trackWidth, RegulatedMotor leftMotor, RegulatedMotor rightMotor, boolean reverse)`
ist der Konstruktor zur Erstellung eines Differential Pilot, der einen Roboter **mit zwei gleich großen** Rädern ansteuert.

Parameter

- `wheelDiameter` – Durchmesser der Räder
- `trackWidth` – Abstand der beiden Räder zueinander
- `leftMotor` – Der linke Motor (meist: Ein Objekt vom Typ EV3LargeRegulatedMotor)
- `rightMotor` – Der rechte Motor (meist: Ein Objekt vom Typ EV3LargeRegulatedMotor)
- `reverse` – **Optionaler** boolescher Parameter zur Angabe der Orientierung der Motoren (siehe oben)

DifferentialPilot`(double leftWheelDiameter, double rightWheelDiameter, double trackWidth, RegulatedMotor leftMotor, RegulatedMotor rightMotor, boolean reverse)`
ist der Konstruktor zur Erstellung eines Differential Pilot, der einen Roboter **mit zwei verschieden großen** Rädern ansteuert.

Parameter

- `leftWheelDiameter` – Durchmesser des linken Rads
- `rightWheelDiameter` – Durchmesser des rechten Rads
- `trackWidth` – Abstand der beiden Räder zueinander
- `leftMotor` – Der linke Motor (meist: Ein Objekt vom Typ EV3LargeRegulatedMotor)
- `rightMotor` – Der rechte Motor (meist: Ein Objekt vom Typ EV3LargeRegulatedMotor)
- `reverse` – Boolesche Variable zur Angabe der Orientierung der Motoren; `true` falls sich der Roboter durch Rotation der Motoren in negative Richtung vorwärts bewegen soll, `false` wenn nicht

Methoden

Nachdem nun ein »DifferentialPilot«-Objekt zur Verfügung steht, kann dieses mit folgenden Methoden angesprochen werden. Die Einheiten der Übergabeparameter entsprechen dabei den Einheiten der im Konstruktor angegebenen Werte für Durchmesser und Abstand der Räder.

arc`(double radius, double angle, boolean immediateReturn)`
lässt den Roboter auf einem Kreis mit dem angegebenen Radius fahren, bis der angegebene Winkel erreicht ist. Ein Wert von 90 Grad entspricht dabei einem Viertelkreis, 180 Grad einem Halbkreis usw. Wird ein positiver Wert für den Radius angegeben, befindet sich der Mittelpunkt des Kreises links vom Roboter. Ist der Wert negativ, befindet sich der Mittelpunkt rechts vom Roboter. Ein Wert von null lässt den Roboter auf der Stelle rotieren. Der Roboter bewegt sich auf dem Radius vorwärts, wenn der angegebene Winkel positiv ist und rückwärts wenn der angegebene Winkel negativ ist.

Parameter

`radius`	– Radius des Kreises
`angle`	– Winkel, den der Roboter auf dem Kreis zurücklegt
`immediateReturn`	– Optionaler Parameter; wenn immediateReturn `true` ist, wird sofort mit der Programmausführung fortgefahren, die Bewegung wird im Hintergrund zu Ende geführt

---

arcBackward`(double radius)`
lässt den Roboter solange rückwärts auf einem Kreis mit dem angegebenen Radius fahren, bis **stop()** aufgerufen wird. Für die Position des Kreismittelpunkts siehe oben (**arc**).

Parameter

`radius` – Radius des Kreises

---

arcForward`(double radius)`
lässt den Roboter solange vorwärts auf einem Kreis mit dem angegebenen Radius fahren, bis **stop()** aufgerufen wird. Für die Position des Kreismittelpunkts siehe oben (**arc**).

Parameter

`radius` – Radius des Kreises

forward ( )
lässt den Roboter solange vorwärts auf einer Linie fahren, bis **stop()** aufgerufen wird.

backward ( )
lässt den Roboter solange rückwärts auf einer Linie fahren, bis **stop()** aufgerufen wird.

isMoving ( )
prüft, ob sich der Roboter zum Zeitpunkt des Methodenaufrufs bewegt und liefert einen entsprechenden booleschen Rückgabewert.

Rückgabewert

`boolean` – `true`, falls sich der Roboter zum Zeitpunkt des Methodenaufrufs bewegt, `false` wenn nicht

isStalled ( )
prüft, ob die Räder des Roboters blockiert sind, während er versucht eine Bewegung auszuführen und liefert einen entsprechenden booleschen Rückgabewert.

Rückgabewert

`boolean` – `true`, falls die Räder blockiert sind, `false` wenn nicht

rotate (`double angle, boolean immediateReturn`)
lässt den Roboter auf seiner Position um den angegebenen Winkel nach rechts oder links rotieren. Ein positiver Wert für den Winkel resultiert in einer Drehung nach links, ein negativer Wert in einer Drehung nach rechts.

Parameter

`angle` – Winkel um den sich der Roboter drehen soll

`immediateReturn` – Optionaler Parameter; wenn immediateReturn `true` ist, wird sofort mit der Programmausführung fortgefahren, die Bewegung wird im Hintergrund zu Ende geführt

rotateLeft `()`
lässt den Roboter auf seiner Position solange nach links drehen, bis **stop()** aufgerufen wird.

---

rotateRight `()`
lässt Roboter auf seiner Position solange nach rechts drehen, bis **stop()** aufgerufen wird.

---

setRotateSpeed `(double rotateSpeed)`
setzt die Rotationsgeschwindigkeit des Roboters in Grad pro Sekunde.

Parameter
`rotateSpeed` – Rotationsgeschwindigkeit in Grad pro Sekunde

---

setTravelSpeed `(double travelSpeed)`
setzt die Bewegungsgeschwindigkeit des Roboters. Einheit ist Einheit des Raddurchmessers pro Sekunde.

Parameter
`travelSpeed` – Bewegungsgeschwindigkeit in Einheit des Raddurchmessers pro Sekunde

---

stop `()`
Stoppt die Bewegung des Roboters.

---

travel `(double distance, boolean immediateReturn)`
bewegt den Roboter auf einer geraden Linie um die angegebene Distanz vorwärts (falls ein positiver Wert für die Distanz angegeben wurde) oder rückwärts (falls ein negativer Wert für die Distanz angegeben wurde).

Parameter
`distance` – Zurückzulegende Distanz
`immediateReturn` – Optionaler Parameter; wenn immediateReturn `true` ist, wird sofort mit der Programmausführung fortgefahren, die Bewegung wird im Hintergrund zu Ende geführt

---

travelArc(`double radius, double distance, boolean immediateReturn`)
bewegt den Roboter auf einem Kreis mit dem angegeben Radius um die angegebene Distanz vorwärts (falls ein positiver Distanzwert übergeben wurde) oder rückwärts (falls ein negativer Distanzwert übergeben wurde). Der Mittelpunkt des Kreises befindet sich auf der linken Seite des Roboters, falls der übergebene Radius positiv ist und auf der rechten Seite, falls der übergebene Radius negativ ist. Der Roboter rotiert auf der Stelle, wenn der angegebene Radius null ist.

Parameter

`radius`	– Radius des Kreises
`distance`	– Auf dem Kreis zurückzulegende Distanz
`immediateReturn`	– Optionaler Parameter; wenn immediateReturn `true` ist, wird sofort mit der Programmausführung fortgefahren, die Bewegung wird im Hintergrund zu Ende geführt

In folgendem Beispielprogramm 8.1 wird gezeigt, wie ein Roboter mit Hilfe des Differential Pilots navigieren kann. Es wird dazu ein Roberta-Robotermodell verwendet, das wieder eine quadratische Bahn abfahren soll (vgl. Programm 7.9). Dem `DifferentialPilot`-Objekt muss dazu lediglich der Radabstand und Raddurchmesser im Konstruktor übergeben werden. Nun kann sich Roberta ganz einfach 30cm vorwärts bewegen und sich danach um 90 Grad auf der Stelle drehen. Dies wird noch drei mal wiederholt, bis das Quadrat abgefahren wurde. Das Programm lässt sich auf jeden anderen Roboter, der von zwei Motoren angetrieben wird, übertragen. Es müssen nur die Werte für den Radabstand (und ggfs. Raddurchmesser) angepasst werden und der Roboter fährt ein Quadrat mit einer Seitenlänge von 30cm.

Programm 8.1: Beispielprogramm zum DifferentialPilot

```
1  import lejos.hardware.motor.EV3LargeRegulatedMotor;
2  import lejos.hardware.port.MotorPort;
3  import lejos.robotics.navigation.DifferentialPilot;
4
5
6  /**
7   * This class describes a Roberta-robot that moves along
        a squared path.
8   * It uses the "DifferentialPilot" class to do so.
9   */
10 public class DifferentialRoberta {
```

```
11      private EV3LargeRegulatedMotor engineR = new
            EV3LargeRegulatedMotor(MotorPort.B);
12      private EV3LargeRegulatedMotor engineL = new
            EV3LargeRegulatedMotor(MotorPort.C);
13      private DifferentialPilot pilot = new
            DifferentialPilot(5.5, 12.5, engineL, engineR);
14
15      private void driveSquare() {
16          for (int i = 0; i < 4; i++) {
17              pilot.travel(30);
18              pilot.rotate(-90);
19          }
20      }
21
22      public static void main(String[] args) {
23          DifferentialRoberta roberta = new
                DifferentialRoberta();
24          roberta.driveSquare();
25      }
26  }
```

Die Klasse »DifferentialPilot« funktioniert für Roboter, deren Antrieb auf zwei unabhänging voneinander angesteuerten Motoren beruht. Jeder Motor treibt dabei direkt ein Rad an, also ohne Übersetzung zum Beispiel durch ein Getriebe, und die beiden Räder liegen auf einer gemeinsamen Achse. Die Lenkbewegung kommt durch die unterschiedlichen Umdrehungsgeschwindigkeiten der beiden Räder zustande. Es gibt aber auch noch ein anderes Fortbewegungskonzept mit zwei Motoren. Dabei ist ein Motor nur für die Vorwärtsbewegung der fest stehenden Hinterräder zuständig, der andere Motor für den Lenkeinschlag eines oder mehrerer Vorderräder. Dieser Lenkmechanismus entspricht im Prinzip dem eines normalen Pkw. Zur Ansteuerung solcher Roboter stellt leJOS die Klasse »SteeringPilot«, ebenfalls aus dem Package »lejos.robotics.navigation«, zur Verfügung. Wir wollen diese Klasse hier allerdings aufgrund des komplexeren Aufbaus von Robotern mit diesem Antriebskonzept nur erwähnen. Für eine detaillierte Beschreibung sei auf die leJOS-API verwiesen.

8.2 Filter

In diesem Abschnitt wird das Konzept der Filter näher erläutert. Sie werden dazu verwendet, die von einem Sensor gelieferten Messwerte auf unterschiedliche Weise zu verändern. Der Sensor liefert die »Samples« (also Messwerte) dabei über einen `SampleProvider`. Die Werte werden vom eingesetzten Filter modifiziert und anschließend als neue »Samples« wieder ausgegeben, weshalb Filter ebenfalls `SamplePro-`

vider darstellen. Mehrere Filter können deswegen auch übereinander genutzt werden, um komplexere Berechnungen zu erlauben. Jeder neue Filter nutzt das Objekt des vorherigen Filters als Parameter usw. Das Endergebnis wird jeweils vom »obersten« Filter-Objekt bezogen. leJOS stellt einige Filter zur direkten Nutzung bereit. Diese befinden sich im Package »lejos.robotics.filter«.

Zur Verwendung eines solchen Filters wird zunächst ein Objekt des gewünschten Filters instanziiert. Dem Konstruktor des Filters müssen dabei der `SampleProvider`, auf den er angewendet werden soll, und die Anzahl der Samples, mit denen er arbeitet (`int bufferSize`), als Parameter übergeben werden. Daraufhin kann durch Aufrufen der Methode `fetchSample(float[] sample, int offset)` des Filters das gefilterte Sample ins angegebene Array geschrieben und weiter verarbeitet werden. Der Filter ruft dafür immer auch die Methode `fetchSample` des zu Grunde liegenden Sensors auf und speichert den so erhaltenen Messwert für die aktuelle und folgende Berechnungen zwischen.

leJOS stellt unter anderem folgende Filter zur Verfügung:

`MeanFilter`: Liefert den Mittelwert über eine bestimmte Anzahl von Samples

`MaximiumFilter`: Liefert das Maximum einer bestimmten Anzahl von Samples

`MinimumFilter`: Liefert das Minimum einer bestimmten Anzahl von Samples

`MedianFilter`: Liefert den Median einer bestimmten Anzahl von Samples

Die Konstruktoren der oben beschriebenen Filter unterscheiden sich lediglich im Namen. **FilterName** ist dabei durch den Namen des gewünschten Filters zu ersetzen.

FilterName`(SampleProvider source, int bufferSize)`

Parameter

`source` – »SampleProvider«-Objekt, auf dem der Filter arbeiten soll; dies kann ein Sample Provider eines Sensors oder ein anderer Filter sein

`bufferSize` – Anzahl der Samples, die der Filter zwischenspeichern soll, um damit zu arbeiten

Integration-Filter

Darüber hinaus gibt es auch noch den `IntegrationFilter`. Dieser liefert das Integral der Messwerte des Sample Providers, auf dem er arbeitet. Er unterscheidet sich von den oben beschriebenen Filtern dadurch, dass ihm keine Anzahl an Messwerten zur Zwischenspeicherung in Form einer Integervariable übergeben wird. Er integriert jeweils zu den Zeitpunkten, an denen seine `fetchSample`-Methode aufgerufen wird. Daher benötigt sein Konstruktor auf nur einen Parameter:

IntegrationFilter `(SampleProvider source)`

Parameter

`source` – »SampleProvider«-Objekt, auf dem der Filter arbeiten soll; dies kann ein Sample Provider eines Sensors oder ein anderer Filter sein

Dieser Filter kann beispielsweise verwendet werden, um die Messwerte eines Gyrosensors von Grad/s in Grad umzuwandeln oder um aus der Winkelgeschwindigkeit eines Motors den tatsächlich rotierten Winkel zu berechnen.

Um die Verwendung eines Filters zu demonstrieren wird nun eine einfache Helligkeitssonde, bestehend aus dem EV3-Stein und einem Farbsensor im »Umgebungslicht«-Modus, programmiert. Beim Drücken der ENTER-Taste nimmt ein `MaximiumFilter` einen Messwert über den Sensor auf und berechnet das Maximum der fünf zuletzt aufgenommenen Messwerte. Dieser Maximalwert wird nach jeder Messung auf dem Display ausgegeben.

Programm 8.2: Beispielprogramm zum MaximumFilter

```
1  import lejos.hardware.Button;
2  import lejos.hardware.lcd.LCD;
3  import lejos.hardware.port.SensorPort;
4  import lejos.hardware.sensor.EV3ColorSensor;
5  import lejos.robotics.SampleProvider;
6  import lejos.robotics.filter.MaximumFilter;
7
8  /**
9   * This class implements a simple ambientlight-probe.
10  * It takes a sample on ENTER and prints the maximum of
11  * the last five measurements on the display.
12  */
13 public class FilterTest {
14
15     public static void main(String[] args) {
16         EV3ColorSensor colorSensor = new EV3ColorSensor(
               SensorPort.S1);
17         SampleProvider spAmbient = colorSensor.
               getAmbientMode();
```

```
18          SampleProvider max = new MaximumFilter(spAmbient,
                5);
19
20          float[] sample = new float[spAmbient.sampleSize()
                ];
21
22          while(Button.ESCAPE.isUp()) {
23              Button.ENTER.waitForPress();
24              max.fetchSample(sample, 0);
25              LCD.drawString("Max. of last 5:", 0, 0);
26              LCD.drawString(Float.toString(sample[0]), 0,
                    1);
27          }
28      }
29  }
```

8.3 Subsumption-Architektur

Die Subsumption-Architektur ist eine reaktive Steuerungsarchitektur für autonome mobile Roboter, die erstmals 1985 von Rodney Brooks und seinen Kollegen am MIT vorgestellt wurde (Brooks 1985). Reaktiv bedeutet in diesem Zusammenhang, dass der Roboter keine interne Repräsentation seiner Umwelt zwischenspeichert, um daraus Handlungsanweisungen abzuleiten. Stattdessen werden alle nötigen Informationen zur Handlungssteuerung direkt aus den aktuellen Sensordaten gewonnen. Dieser Ansatz stand im Gegensatz zur traditionellen Robotikforschung, bei der stets auf eine symbolische Repräsentation der Umwelt zurückgegriffen wurde, und hatte großen Einfluss auf die Entwicklung autonomer Roboter.

In der Subsumption Architektur wird das Gesamtverhalten eines Roboters durch verschiedene »Unterverhalten« gebildet. Diese »Unterverhalten« sind hierarchisch in Form von übereinander liegenden Schichten (engl.: Layer) organisiert, wobei jede Schicht einer bestimmten Aufgabe des Roboters entspricht. Diese Schichten könnten beispielsweise die »Unterverhalten« »Meide andere Objekten« (Schicht 0), »Bewege dich im Raum umher« (Schicht 1), »Entdecke die Umgebung« (Schicht 2) und »Erstelle eine Karte« (Schicht 3) darstellen.

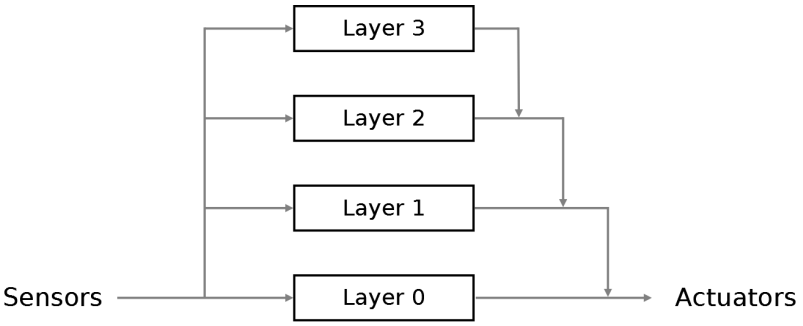


Abbildung 8.1.: Schichten der Subsumption-Architektur (Brooks 1985)

Jede dieser Schichten besteht wiederum aus Modulen, die das gewünschte Verhalten realisieren. Im Fall von Schicht 0 wären das beispielsweise die Module »Lese Sensordaten«, »Berechne Ausweg«, »Fahre berechneten Weg« und »Stoppe wenn Kollision bevorsteht«, die auf entsprechende Sensorinputs mit Anweisungen an die Motoren reagieren. Da die Anordnung der Schichten eine Hierarchie darstellt, können Module höherer Schichten Module tieferer Schichten beeinflussen. Einmal durch sog. »Suppression«-Signale; dabei wird der Input eines Moduls durch das »Suppression«-Signal des hierarchisch höheren Moduls ersetzt. Als nächstes durch sogenannte »Inhibition«-Signale; dabei wird der Output eines Moduls durch das »Inhibition«-Signal eines Moduls höherer Hierarchie unterdrückt. Das Modul niederer Hierarchie kann somit also keine Anweisungen mehr an die Motoren oder an noch tiefere Schichten senden. Auf diese Art und Weise können umfangreiche Verhaltensnetzwerke aufgebaut werden.

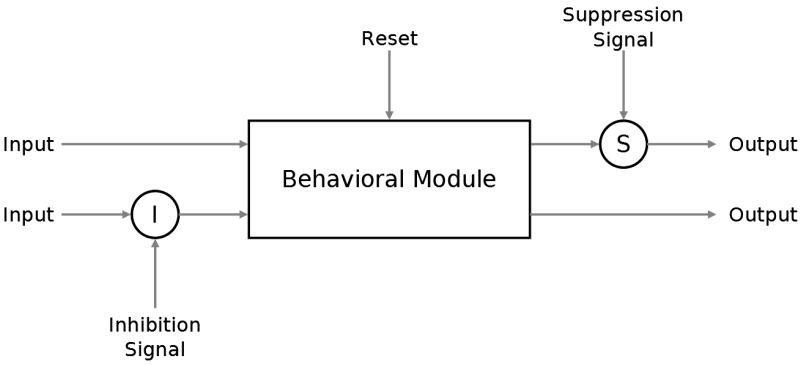


Abbildung 8.2.: Verhaltensmodul der Subsumption-Architektur (Brooks 1985)

Subsumption in leJOS

Die Subsumption-Architektur bietet in der Robotik die Möglichkeit, einen echtzeitfähigen und reaktiven Roboter zu gestalten. Durch die Zentralisierung der Sensorauswertung im EV3-Brick ist es nicht möglich, mit leJOS ein echtzeitfähiges System aufzubauen. Das ist für den Rahmen unserer Anwendungen aber auch nicht nötig. leJOS bietet deswegen dennoch die Möglichkeit, Roboter nach dem Vorbild der Subsumption-Architektur zu programmieren. Dazu wird im Package »lejos.robotics.subsumption« das Interface »Behavior« und die Klasse »Arbitrator« zur Verfügung gestellt. »Behavior« wird benutzt, um verschiedene Verhaltensweisen des Roboters zu beschreiben. »Arbitrator« ist dafür zuständig die Hierarchie zu realisieren, und Verhalten zu aktivieren sowie zu deaktivieren.

Interface Behavior

Mit Hilfe des Interfaces »Behavior« können eigene Klassen definiert werden (vgl. Schnittstellen: Abschnitt 5.1 auf Seite 61). Jede dieser Klassen entspricht dann einem bestimmten Verhalten (engl.: behavior) des Roboters. Denkbar wären zum Beispiel die Verhalten (bzw. Klassen) »Vorwärtsfahren«, »Ausweichen bei erkanntem Hindernis durch Ultraschall«, »Ausweichen bei erkanntem Hindernis durch Taster« und »Notstopp« für einen Roberta-Roboter, der sich in einem Raum bewegen soll. Das Interface »Behavior« gibt dabei nur den Namen und die Art der Methoden vor. Eine abgeleitete Klasse muss dann jede dieser vorgegebenen Methoden implementieren. Beim hier beschriebenen Interface »Behavior« sind das die folgenden drei Methoden:

action ( )
beschreibt die Aktionen, die der Roboter ausführt, wenn dieses Verhalten aktiv wird. Dies kann beispielsweise nur das Abspielen eines Tons sein, oder etwas Komplexeres, wie die Navigation in einem Raum. Es muss berücksichtigt werden, dass die Methode auslaufen muss, sobald die im Folgenden beschriebene Methode **suppress** ( ) aufgerufen wird. Dazu bietet sich beispielsweise die Verwendung einer booleschen Variablen an. Diese wird gesetzt, sobald **suppress** ( ) aufgerufen wird, und in **action** ( ) abgefragt.

---

suppress ( )
Der Aufruf dieser Methode muss das aktuelle Verhalten schnellstmöglich beenden. Eventuell gestartete Threads müssen beendet und die Methode **action** ( ) zum Auslaufen gebracht werden.

---

takeControl ( )
prüft, ob das beschriebene Verhalten die Kontrolle über den Robo-

ter übernehmen und ein entsprechender boolean-Wert zurückgegeben werden sollte. Eine einfache Implementierung könnte beispielsweise die Abfrage eines Touch-Sensors sein, der prüft ob ein Objekt berührt wurde und ein entsprechendes Ausweich-Verhalten aktiviert.

Rückgabewert

`boolean` – Boolesche Variable, die angibt ob das dargestellte Verhalten die Kontrolle über den Roboter übernehmen sollte

Klasse Arbitrator

Die Klasse »Arbitrator« (engl. für Vermittler) ist für die Steuerung und Hierarchisierung von Verhalten zuständig, die mit dem Interface »Behavior« erstellt wurden. Sie hat drei Hauptaufgaben:

- Ermitteln des Verhaltens mit höchster Priorität, dessen **takeControl** `()`-Methode `true` zurückgibt
- Deaktivieren (**suppress** `()`) des aktiven Verhaltens, falls es ein anderes mit höherer Priorität und **takeControl** `()` = `true` gibt
- Aufruf der **action** `()`-Methode des Verhaltens mit höchster Priorität, wenn eine **action** `()`-Methode ausläuft

Ein »Arbitrator« nimmt an, dass ein Verhalten nicht länger aktiv ist, wenn seine **action** `()`-Methode ausgelaufen ist. Deswegen wird er die **suppress** `()`-Methode nur bei denjenigen Verhalten aufrufen, deren **action** `()`-Methoden laufen. Zum Instanziieren eines »Arbitrators« ist folgender Konstruktor zu benutzen:

Arbitrator`(Behavior[] behaviorList, boolean returnWhenInactive)`
instanziiert ein Arbitrator-Objekt mit einer Liste von Behavior-Objekten, die die verschiedenen Verhalten des Roboters darstellen. Der Index des Behavior-Objekts in der Liste entspricht der Priorität des Verhaltens. Das Verhalten mit dem größten Index hat also auch die höchste Priorität. Ist der Arbitrator einmal instanziiert, können die ihm übergebenen Verhalten nicht mehr verändert werden. Nach der Instanziierung muss die Vermittlung des Arbitrators noch mittels **start** `()` gestartet werden.

Parameter

`behaviorList` – Liste der Verhalten zwischen denen der Arbitrator vermittelt

`returnWhenInactive` – Optionaler Parameter; standardmäßig `false`: **start**`()`-Methode läuft nie aus; falls `true`, läuft die **start**`()`-Methode aus, sobald kein Verhalten mehr aktiv ist

8.4 Subsumption-Beispiel: AutonomousRoberta

Wir haben im letzten Kapitel häufig einen Roberta-Roboter verwendet, um grundlegende Funktionen der Komponenten des EV3-Systems, wie Motoren und Sensoren, darzustellen. Mit der Subsumption-Architektur haben wir nun die Möglichkeit einen Roboter mit etwas komplexeren Verhaltensmustern auszustatten. Das Roberta-Grundmodell verfügt über zwei große Motoren zur Fortbewegung, zwei Touch-Sensoren mit Stoßfühlern (die sogenannten »Bumper«) und einen Ultraschallsensor. Damit ist sie perfekt ausgerüstet, um sich autonom in einem Raum umherzubewegen, ihre Bahn blockierende Hindernisse zu entdecken und diesen auszuweichen. Mit diesem Verhalten ist sie vielleicht nicht direkt der hellste Stern am Roboterhimmel, jedoch reagiert sie – ähnlich einem kleinen Insekt – direkt auf die Reize ihrer Umgebung.

Um das eben beschriebene Verhalten mit Hilfe der Subsumption-Architektur von leJOS zu programmieren, werden im Folgenden fünf Klassen verwendet. `MoveForward`, `UltrasonicCollision`, `BumperCollision` sowie `EmergencyStop` implementieren jeweils das Interface `Behavior` und beschreiben die Unterverhaltensweisen der autonomen Roberta. Die Klasse `Roberta` stellt mit ihrer `main`-Funktion das Hauptprogramm dar und beinhaltet den Arbitrator, der für die Hierarchisierung sowie Aktivierung und Deaktivierung der Unterverhaltensweisen zuständig ist.

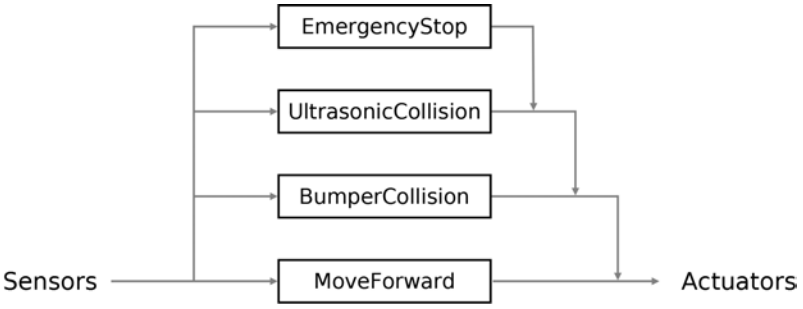


Abbildung 8.3.: Schichten des Subsumption-Beispiels

Es wird in jeder Behavior-Klasse eine boolesche Variable `isActive` verwendet, die anzeigt, wenn ein Verhalten unterdrückt werden soll, also **suppress**`()` aufgerufen wurde. Die **action**`()`-Methoden sind daher so zu implementieren, dass sie direkt auslaufen, sobald `isActive == false` gilt. Es dürfen dort also keine Funktionen verwendet werden, die blockieren, also eine gewisse Zeit brauchen, bis sie auslaufen.

Wir beginnen die Beschreibung mit dem Verhalten niedrigster Priorität: `MoveForward`. Die Implementierung der Klasse ist unter Programm 8.3 angegeben. Sie ist für die reine Vorwärtsbewegung des Roboters zuständig und immer dann aktiv, wenn sie nicht von einer anderen Klasse unterdrückt wird, also kein Hindernis in der Nähe ist und der Not-Stop (ENTER-Taste) nicht betätigt wurde. Ihre **takeControl()**-Methode liefert daher immer `true` zurück. In **action**`()` wird eine `while`-Schleife verwendet, um auf das Umspringen von `isActive` auf `false` zu warten, woraufhin die Motoren direkt gestoppt werden und **action**`()` ausläuft.

Programm 8.3: MoveForward.java

```
1  import lejos.hardware.motor.EV3LargeRegulatedMotor;
2  import lejos.robotics.subsumption.Behavior;
3
4  /**
5   * This class implements the robot moving forward.
6   * This is the lowest priority behavior.
7   */
8  public class MoveForward implements Behavior {
9      private final EV3LargeRegulatedMotor engineR;
10     private final EV3LargeRegulatedMotor engineL;
11     private boolean isActive = false;
12
13     MoveForward(EV3LargeRegulatedMotor motorR,
           EV3LargeRegulatedMotor motorL) {
```

```
14          engineR = motorR;
15          engineL = motorL;
16      }
17
18      public void action() {
19          isActive = true;
20          engineL.setSpeed(Roberta.SPEED);
21          engineR.setSpeed(Roberta.SPEED);
22          engineR.forward();
23          engineL.forward();
24          while (isActive) {}
25          engineR.stop(true);
26          engineL.stop(true);
27      }
28
29      public void suppress() {
30          isActive = false;
31      }
32
33      public boolean takeControl() {
34          return true;
35      }
36  }
```

Das Verhalten nächsthöherer Priorität ist `UltrasonicCollision`; siehe Programm 8.4. Detektiert die autonome Roberta ein Hindernis mit ihrem Ultraschallsensor in einer Entfernung von mindestens 30cm, dreht sie sich zufällig nach rechts oder links, bis sie wieder mindestens 1m freie Strecke vor sich hat (ebenfalls mit dem Ultraschallsensor gemessen). Danach läuft ihre **action** () -Methode aus und das nächste zu aktivierende Verhalten wird vom Arbitrator ermittelt.

Programm 8.4: UltrasonicCollision.java

```
1  import lejos.hardware.motor.EV3LargeRegulatedMotor;
2  import lejos.hardware.sensor.EV3UltrasonicSensor;
3  import lejos.robotics.subsumption.Behavior;
4
5  /**
6   * This class implements the robot's behavior after an
        obstacle is detected by ultrasonic-sensor.
7   * The robot turns randomly to the left or to the right
        until there's at least 1m of free path in front of
        it.
8   */
9  public class UltrasonicCollision implements Behavior {
10      protected final EV3LargeRegulatedMotor engineR;
11      protected final EV3LargeRegulatedMotor engineL;
12      private final EV3UltrasonicSensor uSensor;
13      private float[] distanceSample;
14
15      private boolean isActive = false;
```

```
16
17      UltrasonicCollision(EV3UltrasonicSensor sensor,
            EV3LargeRegulatedMotor motorR,
            EV3LargeRegulatedMotor motorL) {
18          engineR = motorR;
19          engineL = motorL;
20          uSensor = sensor;
21          uSensor.setCurrentMode("Distance");
22          distanceSample = new float[uSensor.sampleSize()];
23      }
24
25      public void action() {
26          isActive = true;
27          //turn to left or right randomly with half the
                motorspeed
28          double rand = Math.random();
29          engineR.setSpeed(Roberta.SPEED * 0.5F);
30          engineL.setSpeed(Roberta.SPEED * 0.5F);
31          if (rand < 0.5) {
32              engineR.forward();
33              engineL.backward();
34          }
35          else {
36              engineL.forward();
37              engineR.backward();
38          }
39          //wait until 1m of path is free or behavior is
                suppressed
40          while (isActive && (distanceSample[0] < 1.0)) {
41              uSensor.fetchSample(distanceSample, 0);
42          }
43          engineL.stop(true);
44          engineR.stop(true);
45      }
46
47      public void suppress() {
48          isActive = false;
49      }
50
51      public boolean takeControl() {
52          uSensor.fetchSample(distanceSample, 0);
53          return (distanceSample[0] < 0.30) ;
54      }
55  }
```

Stößt Roberta mit einem der Bumper gegen ein Hindernis, wird `BumperCollision` aktiviert. Da diese Situation dringender Handlung bedarf, hat nur der Not-Stop noch höhere Priorität. Roberta setzt in diesem Fall ein Stück zurück und dreht sich vom Hindernis weg. Es werden zwei boolesche Variablen verwendet, um zu bestimmen welcher Bumper gegen das Hindernis gestoßen ist und in welche Richtung Roberta

sich drehen muss. Die Implementierung dieses Verhaltens ist in folgendem Programm 8.5 aufgeführt.

Programm 8.5: BumperCollision.java

```
1  import lejos.hardware.motor.EV3LargeRegulatedMotor;
2  import lejos.hardware.sensor.EV3TouchSensor;
3  import lejos.robotics.subsumption.Behavior;
4
5  /**
6   * This class implements the robot's behavior after one
        of its bumpers touches an obstacle.
7   * The robot moves backwards with different motorspeeds (
        on an arc) until the slower motor has turned 180
        degrees.
8   * Depending on which bumper detected the obstacle,
        either the left or the right motor is turning slower
        .
9   * After that the robot faces away from the obstacle and
        can proceed to move forward.
10  */
11 public class BumperCollision implements Behavior {
12     protected final EV3LargeRegulatedMotor engineR;
13     protected final EV3LargeRegulatedMotor engineL;
14     private final EV3TouchSensor tSensorR;
15     private final EV3TouchSensor tSensorL;
16
17     private float[] rBumperSample;
18     private float[] lBumperSample;
19
20     private boolean bLeftBumper = false;
21     private boolean bRightBumper = false;
22     private boolean isActive = false;
23     private final float SPEEDFACTOR = 0.5F;
24     private final int LIMITANGLE = 180;
25
26     BumperCollision(EV3TouchSensor touchR, EV3TouchSensor
           touchL, EV3LargeRegulatedMotor motorR,
         EV3LargeRegulatedMotor motorL) {
27         engineR = motorR;
28         engineL = motorL;
29         tSensorR = touchR;
30         tSensorL = touchL;
31         tSensorR.setCurrentMode("Touch");
32         tSensorL.setCurrentMode("Touch");
33         rBumperSample = new float[tSensorR.sampleSize()];
34         lBumperSample = new float[tSensorL.sampleSize()];
35     }
36
37     private void resetBoolean() {
38         bLeftBumper = false;
39         bRightBumper = false;
40     }
41
```

```
42      public void action() {
43          isActive = true;
44          if (bLeftBumper) {
45              engineL.setSpeed(Roberta.SPEED * SPEEDFACTOR)
                    ;
46          }
47          else if (bRightBumper) {
48              engineR.setSpeed(Roberta.SPEED * SPEEDFACTOR)
                    ;
49          }
50          engineR.resetTachoCount();
51          engineL.resetTachoCount();
52          engineR.backward();
53          engineL.backward();
54          while (isActive && ((engineR.getTachoCount() > -
                LIMITANGLE) || (engineL.getTachoCount() > -
                LIMITANGLE))) {}
55          engineL.stop(true);
56          engineR.stop(true);
57      }
58
59      public void suppress() {
60          isActive = false;
61      }
62
63      public boolean takeControl() {
64          resetBoolean();
65          tSensorR.fetchSample(rBumperSample, 0);
66          tSensorL.fetchSample(lBumperSample, 0);
67          bRightBumper = (rBumperSample[0] == 1.0);
68          bLeftBumper = (lBumperSample[0] == 1.0);
69          return (bRightBumper || bLeftBumper);
70      }
71  }
```

Höchste Priorität hat – wie es bei allen autonomen Robotern der Fall sein sollte – `EmergencyStop`, also der Not-Stop. Er wird ausgelöst, sobald die ENTER-Taste gedrückt wird. Daraufhin stoppt Roberta sofort jegliche Motorbewegung und zeigt ein kleines Menü an. Durch Drücken der RIGHT-Taste kann das autonome Umherfahren und Ausweichen fortgesetzt werden. Drücken der ESCAPE-Taste beendet die Programmausführung gänzlich. Die entsprechende Klasse ist nachfolgend in Programm 8.6 implementiert.

Programm 8.6: EmergencyStop.java

```
1  import lejos.hardware.Button;
2  import lejos.hardware.lcd.LCD;
3  import lejos.hardware.motor.EV3LargeRegulatedMotor;
4  import lejos.robotics.subsumption.Behavior;
5
```

```
6  /**
7   * This class implements the robot's emergency stop-
        function.
8   * When ENTER is pressed, the robot stops its movement
        immediately and waits for further button-input.
9   * RIGHT -> The robot continues it's movement; ESCAPE ->
        The robot's control program exits.
10  */
11 public class EmergencyStop implements Behavior {
12     protected final EV3LargeRegulatedMotor engineR;
13     protected final EV3LargeRegulatedMotor engineL;
14
15     EmergencyStop(EV3LargeRegulatedMotor motorR,
           EV3LargeRegulatedMotor motorL) {
16         engineR = motorR;
17         engineL = motorL;
18     }
19
20     public void action() {
21         engineR.stop(true);
22         engineL.stop(true);
23         int buttonID;
24         LCD.clear();
25         LCD.drawString("Arbitration halted", 0, 0);
26         LCD.drawString("RIGHT to Continue", 0, 1);
27         LCD.drawString("ESCAPE to Exit", 0, 2);
28         do {
29             buttonID = Button.waitForAnyPress();
30         }
31         while (buttonID != Button.ID_RIGHT && buttonID !=
               Button.ID_ESCAPE);
32         if (buttonID == Button.ID_ESCAPE) {System.exit(1)
               ;}
33         else {
34             LCD.clear();
35             LCD.drawString("Arbitrating", 0, 0);
36         }
37     }
38
39     public void suppress() {
40     }
41
42     public boolean takeControl() {
43         return (Button.ENTER.isDown());
44     }
45 }
```

Die Klasse `Roberta` beinhaltet neben dem Hauptprogramm und dem Arbitrator auch Variablen für die Sensoren und Motoren des Roberta-Roboters. Diese werden den anderen Behavior-Klassen, die ebenfalls darauf zugreifen müssen, als Konstruktorparameter übergeben. Bei der Speicherung der Behavior-Klassenobjekte in das Array `behaviorList[]` ist die Hierarchie in Form des Arrayindexes gut zu erkennen.

Jeder Eintrag des Arrays (also jedes Objekt) bildet damit genau ein Layer der Subsumption-Architektur.

Da so gut wie die gesamte Funktion in die Verhaltensklassen ausgelagert wurde, ist das Hauptprogramm sehr kurz. Es muss lediglich ein Objekt der Klasse `Roberta` instanziiert und der dazugehörige Arbitrator gestartet werden.

Programm 8.7: Roberta.java

```
1  import lejos.hardware.lcd.LCD;
2  import lejos.hardware.motor.EV3LargeRegulatedMotor;
3  import lejos.hardware.port.MotorPort;
4  import lejos.hardware.port.SensorPort;
5  import lejos.hardware.sensor.EV3TouchSensor;
6  import lejos.hardware.sensor.EV3UltrasonicSensor;
7  import lejos.robotics.subsumption.Arbitrator;
8  import lejos.robotics.subsumption.Behavior;
9
10 /**
11  * This class describes a Roberta-robot that can move
        around in a room while avoiding obstacles.
12  * It uses four Behavior-classes as seen in its
        constructor. MoveForward has lowest priority,
        EmergencyStop has highest priority.
13  * The robot uses either its ultrasonic-sensor or its
        bumpers to detect an obstacle.
14  */
15 public class Roberta {
16     EV3LargeRegulatedMotor engineR = new
           EV3LargeRegulatedMotor(MotorPort.B);
17     EV3LargeRegulatedMotor engineL = new
           EV3LargeRegulatedMotor(MotorPort.C);
18     EV3UltrasonicSensor uSensor = new EV3UltrasonicSensor
           (SensorPort.S4);
19     EV3TouchSensor bumperR = new EV3TouchSensor(
           SensorPort.S1);
20     EV3TouchSensor bumperL = new EV3TouchSensor(
           SensorPort.S2);
21
22     Arbitrator arby;
23     Behavior[] behaviorList = new Behavior[4];
24
25     public final static int SPEED = 360;
26
27     Roberta() {
28         behaviorList[0] = new MoveForward(engineR,
               engineL);
29         behaviorList[1] = new UltrasonicCollision(uSensor
               , engineR, engineL);
30         behaviorList[2] = new BumperCollision(bumperR,
               bumperL, engineR, engineL);
```

```
31          behaviorList[3] = new EmergencyStop(engineR,
                engineL);
32          arby = new Arbitrator(behaviorList);
33      }
34
35      public static void main(String[] args) {
36          Roberta autonomousRoberta = new Roberta();
37          LCD.drawString("Arbitrating", 0, 0);
38          autonomousRoberta.arby.start();
39      }
40  }
```

8.5 leJOS-Utility: Stopwatch

Die Klasse »Stopwatch« aus dem Package »lejos.utility« gehört zwar nicht zu den leJOS-Robotikklassen, soll aufgrund ihrer Praktikabilität hier aber trotzdem kurz erläutert werden. Dabei handelt es sich um eine einfach zu benutzende Stoppuhr, die für verschiedene Zeitanwendungen genutzt werden kann. Ein »Stopwatch«-Objekt kann mit dem parameterlosen Konstruktor **Stopwatch** `()` instanziiert werden. Die Zeitmessung startet unmittelbar. Um auf die gemessene Zeit zuzugreifen gibt es folgende zwei Methoden:

elapsed `()`
gibt die Anzahl der seit dem Start der Stopwatch verstrichenen Millisekunden als Integer zurück.

Rückgabewert
`int` – Anzahl der verstrichenen Millisekunden

---

reset `()`
setzt die Stopwatch auf null zurück. Die Messung der verstrichenen Millisekunden wird unmittelbar nach dem Aufruf dieser Methode fortgesetzt.

Die Verwendung der Klasse `Stopwatch` wird im Beispielprogramm 8.8 demonstriert. Mit ihrer Hilfe wird eine klassische Stoppuhr programmiert. Beim ersten Drücken der ENTER-Taste wird die Stoppuhr gestartet und dies auf dem Display ausgegeben. Die wiederholte Betätigung der ENTER-Taste stoppt die Uhr und gibt die verstrichene Zeit auf dem Display aus. Das Programm kann daraufhin durch Drücken der ESCAPE-Taste verlassen werden.

Programm 8.8: StopwatchTester.java

```
1  import lejos.hardware.Button;
2  import lejos.hardware.lcd.LCD;
3  import lejos.utility.Stopwatch;
4
5  /**
6   * This class describes a simple stopwatch-program.
7   * The ENTER-Button starts and stops the measurement.
8   * The passed time is the printed on the EV3-display.
9   */
10 public class StopwatchTester {
11     private Stopwatch timer = new Stopwatch();
12
13     private void startMeasuring() {
14         int time;
15         timer.reset();
16         LCD.drawString("Timer started", 0, 1);
17
18         Button.ENTER.waitForPress();
19         time = timer.elapsed();
20         LCD.drawString("Timer stopped", 0, 2);
21
22         LCD.drawString("", 0, 3);
23         LCD.drawString("Time passed:", 0, 4);
24         LCD.drawString(time + " milliseconds", 0, 5);
25         LCD.drawString((time / 1000.0) + " seconds", 0,
               6);
26     }
27
28     public static void main(String[] args) {
29         StopwatchTester watch = new StopwatchTester();
30         LCD.drawString("ENTER to start", 0, 0);
31         Button.ENTER.waitForPress();
32
33         watch.startMeasuring();
34
35         LCD.drawString("ESCAPE to exit", 0, 7);
36         Button.ESCAPE.waitForPress();
37     }
38 }
```

Die Ausgabe des Programms auf dem EV3-Display ist in Abbildung 8.4 auf der nächsten Seite zu sehen.

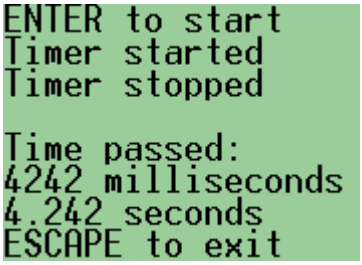


Abbildung 8.4.: Displayausgabe des Stopwatch-Programms

Tipps, Tricks und leJOS-Experiment

In diesem Kapitel werden Tipps und Tricks beschrieben, die den Programmierer in der Praxis mit leJOS und dem EV3 unterstützen. Neben praktischen Tastenkombinationen werden auch gängige Probleme bei der Verbindung des EV3 mit dem Computer behandelt.

9.1 Tipps und Tricks

Tastenkombinationen

Es kann aus verschiedenen Gründen vorkommen, dass ein laufendes Programm nicht beendet wird und der EV3 auf keine Benutzereingaben mehr reagiert. Oft handelt es sich dabei um einen einfachen Programmierfehler, wenn zum Beispiel die Abbruchbedingung einer Schleife nie erfüllt wird, oder der EV3 hat sich schlicht aufgehangen und reagiert nicht mehr. Die intuitive Lösung in einem solchen Fall wäre den Akku zu entfernen, wieder einzusetzen und damit den EV3 neu zu starten. Der EV3 ist aber meist in einen Roboter eingebaut, was das entfernen des Akkus schwierig machen kann. Ein kompletter Neustart des EV3 ist bei einem Programm, das auf Grund eines Programmierfehlers nicht beendet wird, aber auch gar nicht nötig.

- **Manuelles Beenden eines Programms:** Ist der EV3 nicht abgestürzt, so kann jedes laufende Programm durch Gedrückthalten der Tasten `ENTER` und `DOWN` manuell beendet werden. Der EV3 kehrt daraufhin ins Hauptmenü zurück und startet nicht neu.
- **Manuelles Neustarten des EV3:** Ist der EV3 – aus welchem Grund auch immer – abgestürzt und reagiert auf keinerlei Eingaben, so kann er durch Gedrückthalten der Tasten `ESCAPE`, `ENTER` und `DOWN` manuell neu gestartet werden.

Verbindung mit dem EV3

Beim Verbinden des Computers mit dem EV3 über USB, wie in Anhang A beschrieben, können Probleme auftreten. In seltenen Fällen kommt es vor, dass eine vorher problemlos funktionierende Verbindung plötzlich nicht mehr funktioniert. Bevor aufwändigere Maßnahmen ergriffen werden, ist es ratsam, zunächst das USB-Kabel aus- und wieder einzustecken sowie ggfs. den EV3 neu zu starten. Im Folgenden werden Maßnahmen beschrieben, die darüber hinaus die Verbindung mit dem EV3 verbessern können.

- **Eclipse die IP des EV3 mitteilen:** Die IP des EV3 wird im Hauptmenü auf dem Display des EV3 unter seinem Namen angezeigt. Diese IP (standartmäßig: 10.0.1.1) kann in Eclipse unter Window → Preferences in der Kategorie »leJOS EV3« nach setzen des Häckchens vor »Connect to named Brick« im Feld »Name:« eingetragen werden. Nach Bestätigen mittels OK kann die Verbindung erneut überprüft werden.
- **Der Netzwerkschnittstelle eine statische IP zuweisen:** Für die Verbindung des EV3 mittels USB wird eine Netzwerkschnittstelle emuliert, das sog. »RNDIS/Ethernet Gadget«. Dieser Schnittstelle wird auch automatisch eine IP-Adresse zugewiesen (in der Regel 10.0.1.10). Es kann allerdings vorkommen, dass eine andere IP-Adresse zugewiesen wird, was eine erfolgreiche Verbindung verhindert. Die zugewiesene IP kann durch Ausführen des Befehls `ipconfig /all` in der Windows-Kommandozeile unter dem Eintrag »RNDIS/Ethernet Gadget« überprüft werden. Falls die vorhandene IP-Adresse nicht mit der standardmäßig verwendeten IP übereinstimmt, muss der Schnittstelle diese IP manuell zugewiesen werden. Dazu sind unter Windows die Netzwerkadaptereinstellungen der RNDIS-Schnittstelle anzupassen. Unter den Eigenschaften der Schnittstelle muss das IPv4-Protokoll ausgewählt und dort folgende Werte eingetragen werden:
  - IP-Adresse: 10.0.1.10
  - Subnetzmaske: 255.255.255.0
  - Bevorzugter DNS-Server: 10.0.1.1

Compiler Compliance Level

In Versionen, älter als 0.9.0, benötigt leJOS das Compiler Compliance Level 1.7. In der Regel sollte diese Einstellung beim Anlegen eines leJOS-Projekts in Eclipse automatisch erfolgen. Das Compliance Level kann für ein Projekt durch Rechtsklick auf das Projekt und Auswählen von Properties im darauffolgenden Fenster unter »Java Compiler« verändert werden.

9.2 leJOS-Experiment

An dieser Stelle wird ein weiteres Beispiel für eine praktische Anwendung des LEGO EV3-Systems mit leJOS vorgestellt: der »Roberta-EV3CubeSolver«. Dabei handelt es sich um einen Roboter, der in der Lage ist sogenannte »Rubik's Cubes« (auch »Zauberwürfel« genannt)

zu lösen. Die Konstruktion ist in Abbildung 9.1 dargestellt[1]. Im Folgenden wird kurz auf den Aufbau und die Funktionsweise eingegangen.

Die Bauanleitung und eine ausführliche Dokumentation können im Roberta-Portal unter www.roberta-home.de/javaband-ev3 heruntergeladen werden.

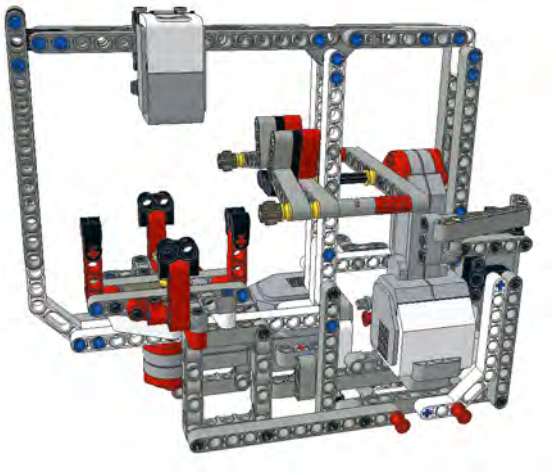


Abbildung 9.1.: Der Roberta-EV3CubeSolver ohne Verkabelung und EV3-Brick

Roberta-EV3CubeSolver

Der »Roberta-EV3CubeSolver« kann komplett aus einem LEGO MINDSTORMS Education-Set aufgebaut werden und verwendet zum Lösen des Würfels die drei enthaltenen Motoren. Die beiden großen Servomotoren sind für die Bewegung des sogenannten »Tischs« (auf dem der Würfel ruht; siehe Abbildung 9.3 auf der nächsten Seite) und der sogenannten »Gabel« (mit der die ihr zugewandten Seite des Würfels gedreht wird; siehe Abbildung 9.2 auf der nächsten Seite) zuständig. Der Servomotor medium ist für die Bewegung der »Gabel« in Richtung Würfel und wieder zurück zuständig.

1 Sie beruht auf dem Modell eines NXT-Cubesolvers von »Copper Dragon«, welches unter www.youtube.com/watch?v=1ty5MxgaGak angsehen werden kann

Abbildung 9.2.: Gabel

Abbildung 9.3.: Tisch

Darüber hinaus wird der Farbsensor verwendet um die Farben der verschiedenen Elemente des Würfels zu detektieren. Bevor der Lösungsalgorithmus angewendet werden kann muss der Würfel komplett »gescannt« werden. Aufgrund des Aufbaus ist dies aber immer nur für die Oberseite des Würfels möglich. Deswegen muss der Würfel dabei nach einem bestimmten Muster verdreht werden, sodass sich jedes Element einmal oben befunden hat.

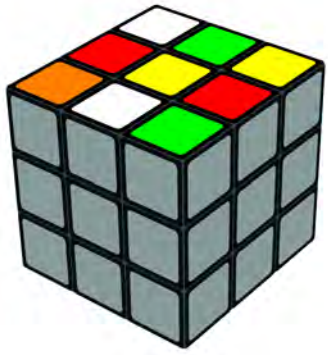


Abbildung 9.4.: Abgescannte Oberseite des Rubik's Cube

Ist der Scanvorgang abgeschlossen befindet sich eine interne Repräsentation des gescannten physikalischen Würfels in Form eines mehrdimensionalen Arrays im Speicher des Cubesolvers. Mit Hilfe von zwei Indizes werden dabei die Farbwerte der einzelnen Elemente abgespeichert. Der erste Index beschreibt die Seite des Würfels, auf dem sich das Element befindet, der zweite ist für die Adressierung des Elements zuständig. Auf einer Seite werden dabei die Elemente von oben links im Uhrzeigersinn durchnummeriert. Das mittlere Element einer Seite ist

fest und braucht nicht nummeriert werden. Eine mögliche Konfiguration des Arrays, als Matrix dargestellt, könnte wie folgt aussehen:

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								

Tabelle 9.1.: Eine mögliche Konfiguration des Rubik's Cube

Die Zeilen entsprechen dabei den Seiten und die Spalten dem jeweiligen Element. Mit diesem Abbild kann dann die Lösung berechnet werden, die zu guter letzt auf dem physikalischen Würfel ausgeführt wird. Dabei verwendet der Roboter einen Algorithmus nach Schichten, der so auch von einem Menschen angewendet werder könnte. Nach dem Beenden des Vorgangs befindet sich der Würfel gelöst im »Roberta-EV3CubeSolver« und kann zur erstaunten Betrachtung entnommen werden.

Exemplarisch ist in Programm(ausschnitt) 9.1 die `main`-Methode des »Roberta-EV3CubeSolver« zu sehen. Die Codekommentare zeigen die Aufteilung des Hauptprogramms in die Abschnitte »Scannen«, »Berechnung der Lösung« und »Ausführen der Lösung«.

Programm 9.1: main-Methode des Roberta-EV3CubeSolver

```
1       public static void main (String[] args) {
2           int scanTime, searchTime, applyTime;
3
4           MotorController motors = new MotorController();
5           CubeScanner scanner = new CubeScanner(motors);
6           RubiksCube cube = new RubiksCube();
7           CubeAlgorithm solver = new CubeAlgorithm();
8           Stopwatch timer = new Stopwatch();
9           motors.init();
10
11          LCD.drawString("Start Scanning?",0,0);
12          Button.ENTER.waitForPress();
13
14          //Scanning starts here
15          timer.reset();
16          scanner.scanCube(cube);
17          while (cube.completeIntegrity() == false) {
18              LCD.clear();
```

```
19              LCD.drawString("Scan Failure", 0, 0);
20              LCD.drawString("Rescan?", 0, 1);
21              Button.ENTER.waitForPress();
22              timer.reset();
23              motors.rotate();
24              cube.resetNull();
25              scanner.scanCube(cube);
26          }
27          scanTime = timer.elapsed();
28
29          cube.printEV3();
30          LCD.drawString("Start Search?", 0, 7);
31          Button.ENTER.waitForPress();
32
33          //Calculation of solution starts here
34          cube.setRecording(true);
35          timer.reset();
36          solver.firstLevelEdges(cube);
37          cube.shortenSolution();
38          solver.firstLevelCorners(cube);
39          cube.shortenSolution();
40          solver.secondLevelEdges(cube);
41          cube.shortenSolution();
42          solver.orientLLEdges(cube);
43          cube.shortenSolution();
44          solver.permuteLLCorners(cube);
45          cube.shortenSolution();
46          solver.orientLLCorners(cube);
47          cube.shortenSolution();
48          solver.permuteLLedges(cube);
49          cube.shortenSolution();
50          searchTime = timer.elapsed();
51
52          LCD.clear();
53          LCD.drawString("Solution found", 0, 0);
54          LCD.drawString(""+cube.solutionIndex, 0, 1);
55          LCD.drawString("Apply?", 0, 2);
56          Button.ENTER.waitForPress();
57
58          //Solution is applied to cube here
59          timer.reset();
60          motors.maneuver(cube.solution, cube.solutionIndex
                );
61          applyTime = timer.elapsed();
62
63          //Times are displayed
64          LCD.clear();
65          LCD.drawString("Scan: "+ scanTime/1000.0F, 0, 0);
66          LCD.drawString("Search: "+ searchTime/1000.0F, 0,
                 1);
67          LCD.drawString("Apply: "+ applyTime/1000.0F, 0,
                2);
68
69          Button.ENTER.waitForPress();
70      }
```

Installation

Um den EV3 mit leJOS und Java programmieren zu können, müssen die in diesem Kapitel beschrieben Komponenten installiert werden. Diese Installationsanleitung kann Schritt für Schritt am eigenen Rechner nachvollzogen werden. Damit es direkt losgehen kann sollte kurz überprüft werden ob die benötigte Hardware vorhanden ist:

- Ein LEGO MINDSTORMS EV3
- Eine microSD-Karte (2 GB oder größer, max. 32 GB)
- Ein SD-Karten-Lesegerät (dieses ist oft im PC oder Laptop integriert, ansonsten als USB-Stick bzw. externes Lesegerät erhältlich)
- Ein »microSD auf SD«-Adapter (meist bei microSD-Karten dabei)
- Ein USB-WLAN-Adapter **oder**
- Das USB Kabel aus dem EV3-Set

A.1 Installation leJOS und Java

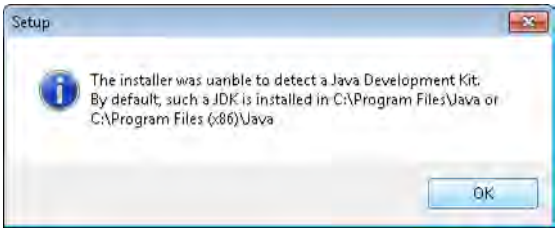
leJOS und Java auf einem Computer mit einem Windows Betriebssystem (hier Windows 7) zu installieren ist Dank des Installationsprogramms von leJOS vergleichsweise einfach. Zunächst muss dieses von http://www.lejos.org unter der Rubrik **Downloads** aus dem Ordner **0.9.0-beta** heruntergeladen werden[1]. Es trägt den Namen:

- `leJOS_EV3_0.9.0-beta_win32_setup.exe`

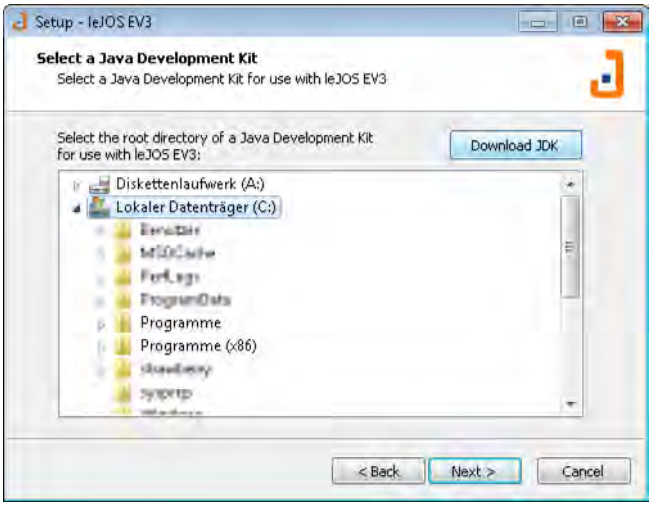
Nach dem Start, durch Doppelklick auf das Programm, muss die Installation dem Windows System eventuell erst erlaubt werden, in dem die Meldung »Der Herausgeber konnte nicht verifiziert werden. Möchten Sie diese Software ausführen?« mit Ausführen und anschließend mit Ja bestätigt wird. Folgender Bildschirm sollte nun erscheinen:

1 Folgender link führt direkt zur Datei: http://sourceforge.net/projects/lejos/files/lejos-EV3/0.9.0-beta/leJOS_EV3_0.9.0-beta_win32_setup.exe/download

Die Installation wird mit einem Klick auf Next gestartet. Die leJOS Installation sucht nun automatisch nach einem 64-Bit Java Development Kit (unterstützt aber auch 32-Bit Versionen). Wenn bereits eins auf dem Computer installiert ist, kann die folgende Anleitung der Java Installation bis »Umgebungsvariablen« auf Seite 153 übersprungen werden, ansonsten erscheint folgende Meldung:

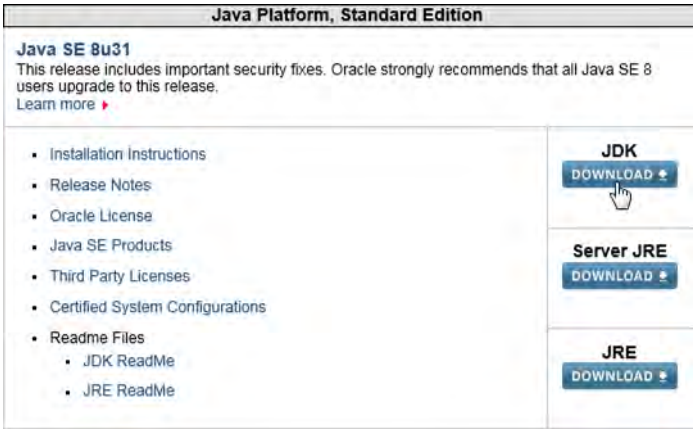


Nach einem Klick auf OK gelangt man zu folgendem Fenster:

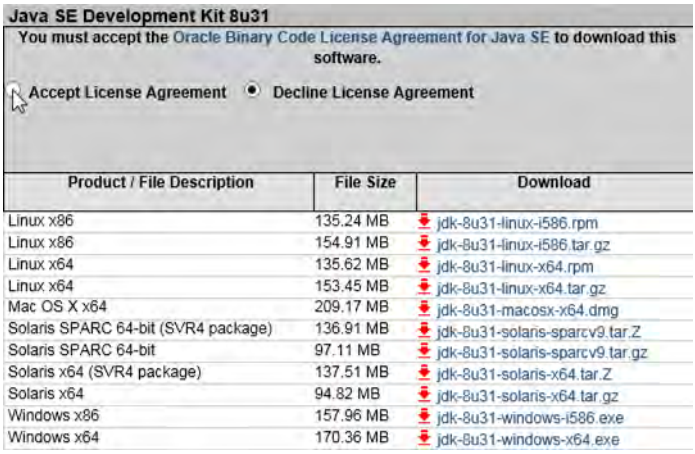


Installation des Java Development Kit

Ein Klick auf Download JDK führt zur Website von Oracle, auf der ein passendes JDK heruntergeladen werden kann.



Hier geht es nun weiter mit einem Klick auf Download unter **JDK**.



Java SE Development Kit 8u31

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux x86	135.24 MB	jdk-8u31-linux-i586.rpm
Linux x86	154.91 MB	jdk-8u31-linux-i586.tar.gz
Linux x64	135.62 MB	jdk-8u31-linux-x64.rpm
Linux x64	153.45 MB	jdk-8u31-linux-x64.tar.gz
Mac OS X x64	209.17 MB	jdk-8u31-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	136.91 MB	jdk-8u31-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	97.11 MB	jdk-8u31-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	137.51 MB	jdk-8u31-solaris-x64.tar.Z
Solaris x64	94.82 MB	jdk-8u31-solaris-x64.tar.gz
Windows x86	157.96 MB	jdk-8u31-windows-i586.exe
Windows x64	170.36 MB	jdk-8u31-windows-x64.exe

Nun ist in der Rubrik »Java SE Development Kit 8u31« die Zeile »Windows x86« für ein 32-Bit Windows-System oder »Windows x64« für ein 64-Bit Windows-System auszuwählen.

Ob es sich bei dem verwendeten Rechner um ein 32- oder 64-Bit System handelt kann in Windows 7 durch Klicken von »Start« → »Systemsteuerung« → »System und Sicherheit« → »System« im Eintrag »Systemtyp« nachgesehen werden.

In der Spalte **Download** befinden sich die Downloadlinks für die Installationsprogramme. Bevor per Klick auf den entsprechenden Link der Download gestartet wird, muss »Accept License Agreement« angeklickt werden. Sobald der Download beendet ist, wird die Java-Installation per Doppelklick auf das Installationsprogramm gestartet.

Die Versionsbezeichnungen können sich nach Updates von Oracle ändern und stimmen dann nicht mehr mit den hier dargestellten Abbildungen überein. Die Installation verläuft in diesem Fall aber analog.

Die Installation des »Java SE Development Kit« wird per Klick auf Next gestartet.

An dieser Stelle kann der Installationspfad des »JDK« verändert werden. Wir empfehlen die Voreinstellung beizubehalten und mit Next fortzufahren. Die Wartezeit kann mit der Beobachtung des schönen, grünen Fortschrittbalkens überbrückt werden.

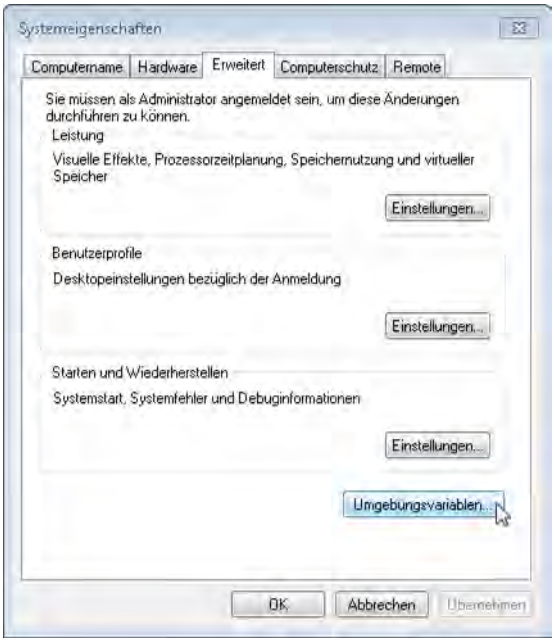
Der Installationspfad für die »JRE« braucht ebenfalls nicht verändert werden. Mit Weiter wird die Angabe bestätigt und mit der Installation fortgefahren.

Nach erfolgreicher Installation beendet ein Klick auf Close das Installationsprogramm.

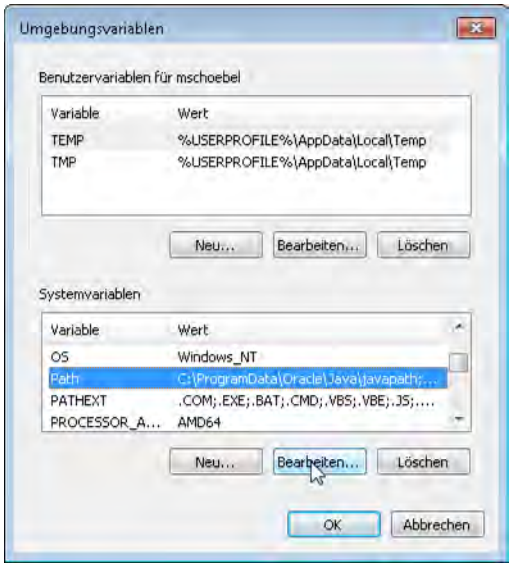
Umgebungs-variablen

Nun müssen die Umgebungsvariablen »PATH« und »JAVA_HOME« gesetzt werden. Damit wird sichergestellt, dass die eben installierten Java Dateien auch von anderen Programmen im System gefunden werden können. Dies funktioniert wie folgt:

Über »Start« → »Systemsteuerung« → »System und Sicherheit« → »System« → »Erweiterte Systemeinstellungen« gelangt man zum Fenster der Systemeigenschaften.



Unter »Erweitert« gelangt man über den Button Umgebungsvariablen… zu dem Fenster in dem die Umgebungsvariablen gesetzt werden können (siehe nächste Abbildung).



Unter »Systemvariablen« muss die Variable »PATH« per Klick markiert werden. Anschließend öffnet Bearbeiten… ein neues, kleineres Fenster. Im Feld »Wert der Variablen« befinden sich schon einige Einträge. Hinter dem letzten Eintrag ist ein Strichpunkt, gefolgt vom Pfad zum `bin`-Ordner im Verzeichnis des »JDK« einzutragen. Konkret muss also hinter den letzten Eintrag folgendes eingefügt werden.

Für 32-Bit-Systeme:

- `;C:\Program Files (x86)\Java\jdk1.8.0_31\bin`

Für 64-Bit-Systeme:

- `;C:\Program Files\Java\jdk1.8.0_31\bin`

Danach muss noch die »JAVA_HOME«-Variable erstellt werden. Dazu öffnet neu… unter »Systemvariablen« wieder ein kleines Fenster, diesmal aber mit leeren Feldern. In »Name der Variablen« wird `JAVA_HOME` eingetragen. Das Feld »Wert der Variablen« wird wie folgt gefüllt.

Für 32-Bit-Systeme:

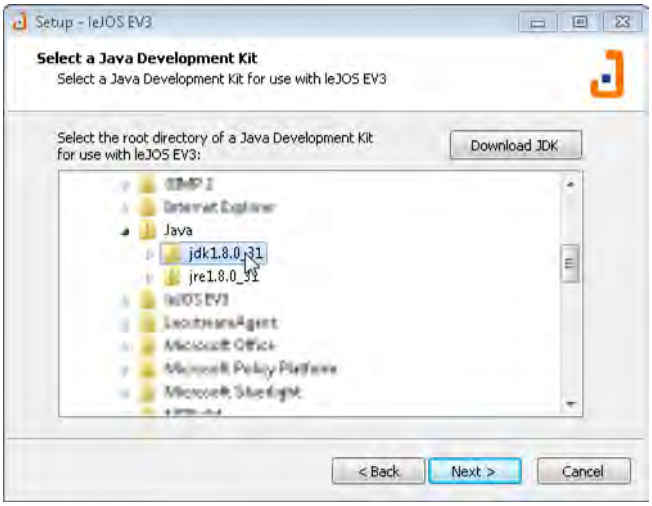
- `C:\Program Files (x86)\Java\jdk1.8.0_31`

Für 64-Bit-Systeme:

- `C:\Program Files\Java\jdk1.8.0_31`

Installation leJOS

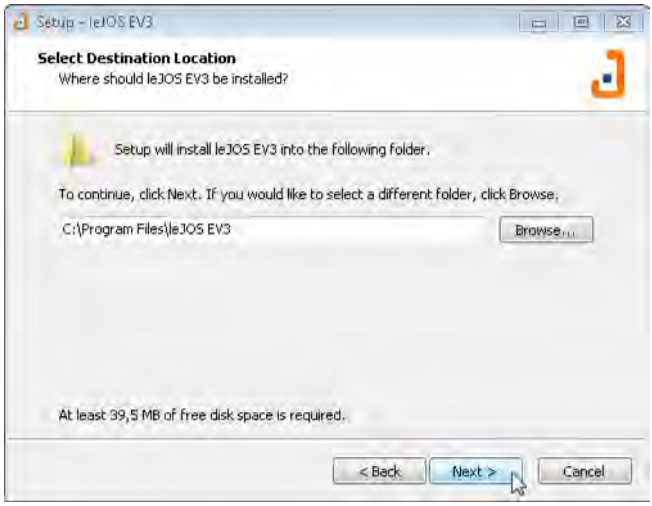
Nachdem die Java-Installation nun vollständig abgeschlossen ist, folgt die Installation von leJOS. Falls das Installationsprogramm nicht mehr geöffnet ist, muss es durch einen Doppelklick auf die zu Beginn heruntergeladene `leJOS_EV3_0.9.0-beta_win32_setup.exe` erneut gestartet werden. Ansonsten befinden wir uns bei der Angabe des Java-Installationsverzeichnis:



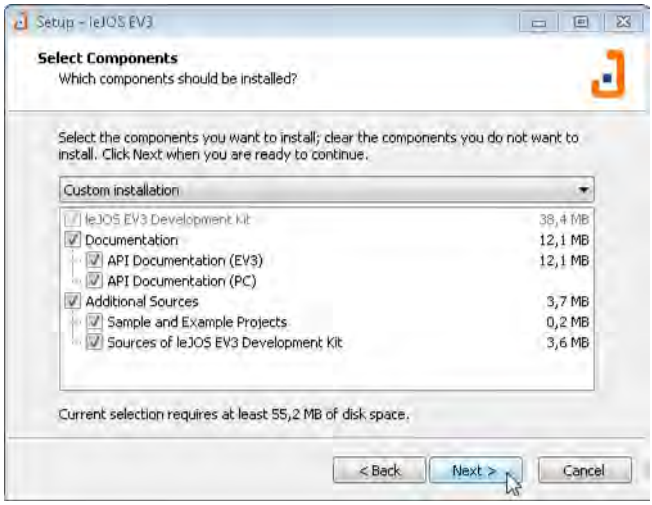
Unter

- `C:\Program Files (x86)\Java` bei einer 32-Bit Java-Installation
- `C:\Program Files\Java` bei einer 64-Bit Java-Installation

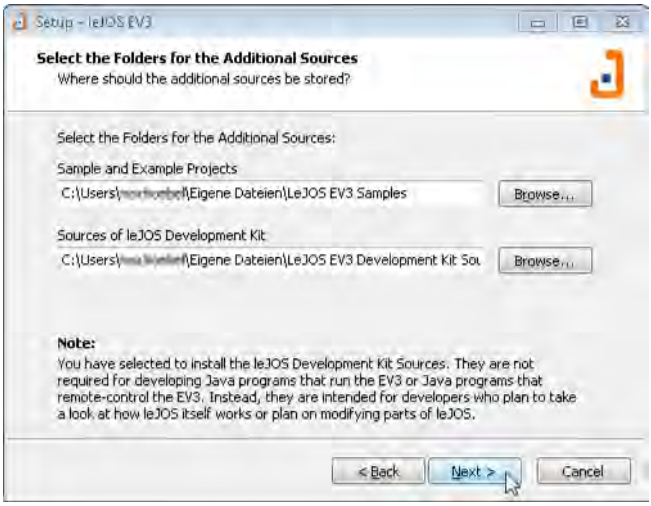
kann nun den Ordner »jdk1.8.0_31« gefunden werden. Dieser ist anzuwählen und mittels Next zu bestätigen.



Der Installationspfad für leJOS sollte nicht verändert werden. Es geht mit Klick auf Next weiter.

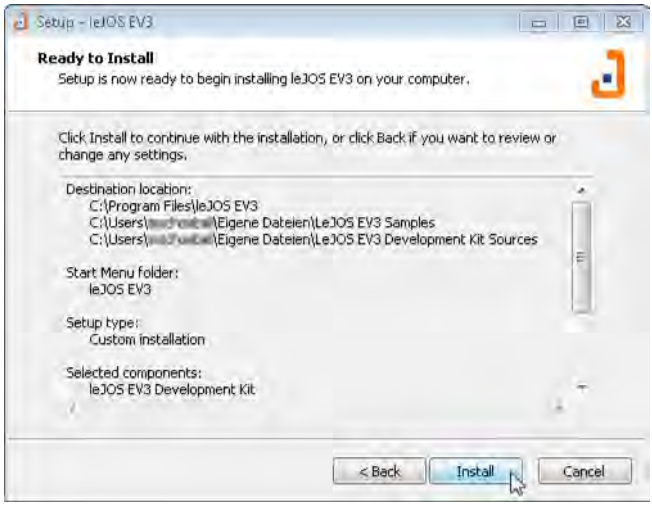


Damit am Ende nichts fehlt, ist hier darauf zu achten, dass alle Häkchen gesetzt sind. Next bestätigt die Auswahl und führt zu folgendem Fenster:



leJOS möchte den Sourcecode und die Beispiele im Benutzerverzeichnis ablegen. Hier kann auch ein anderes Verzeichnis nach Belieben ausgewählt werden. Es geht mit Next weiter.

Hier sind keine Änderungen nötig. Nochmal Next.



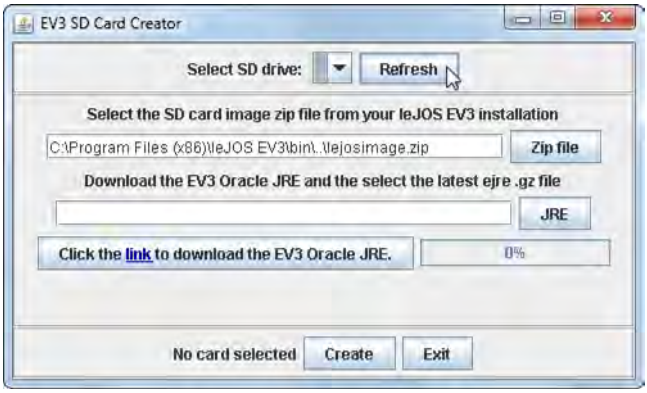
Nachdem nun alle Angaben nochmals geprüft wurden kann die Installation per Klick auf Install gestartet werden.

Die Installation von leJOS war erfolgreich. Mit dem Häckchen bei »Launch EV3SDCard utility« wird nach Klick auf Finish ein kleines Programm gestartet mit dem im nächsten Abschnitt die leJOS-SD-Karte zur Benutzung im EV3 erstellt wird.

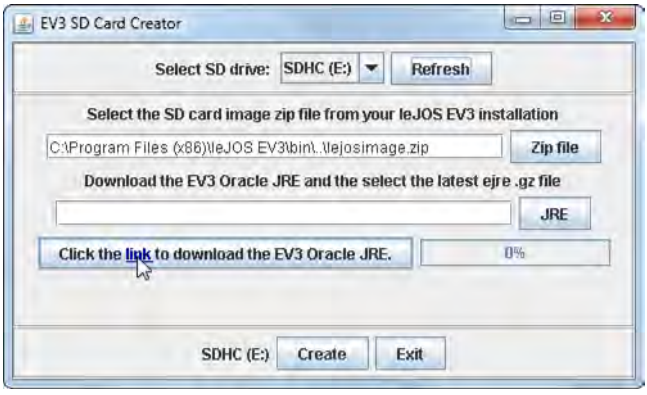
A.2 microSD-Karte für den EV3 erstellen

Sollte das SD-Karten-Programm nach der leJOS-Installation nicht automatisch gestartet worden sein, kann man es manuell im Verzeichnis `C:\Program Files\leJOS EV3\bin` durch Doppelklick auf »ev3sdcard« aufrufen.

Zunächst muss die microSD-Karte (über den Adapter) mit dem Computer verbunden werden.



Nach einem Klick auf Refresh ist das Laufwerk mit der microSD-Karte unter »Select SD drive:« auszuwählen.



Das Programm hat bereits das `lejosimage.zip` gefunden, in diesem Feld brauchen also keine Veränderungen mehr vorgenommen werden. Es wird noch eine Java-Version für Embedded-Systeme benötigt. Mit einem Klick auf link gelangt man im Browser zum Download-Bereich von Oracle.

Als erstes gibt es eine Meldung für Cookies. Diese kann mit Ask me later » geschlossen werden. Daraufhin muss zunächst die Lizenzbedingung durch Anklicken von »Accept License Agreement« bestätigt werden.


Oracle Java SE Embedded

You must accept the OTN License Agreement to download this software.
Accept License Agreement | Decline License Agreement

Oracle Java SE Embedded version 8

Product / File Description	File Size	Download
ARMv5 Linux - Headless EABI, SoftFP ABI, Little Endian[1]	96 MB	ejdk-8-fcs-b132-linux-arm-sflt-03_mar_2014.tar.gz

Oracle Java SE Embedded version 7 Update 60

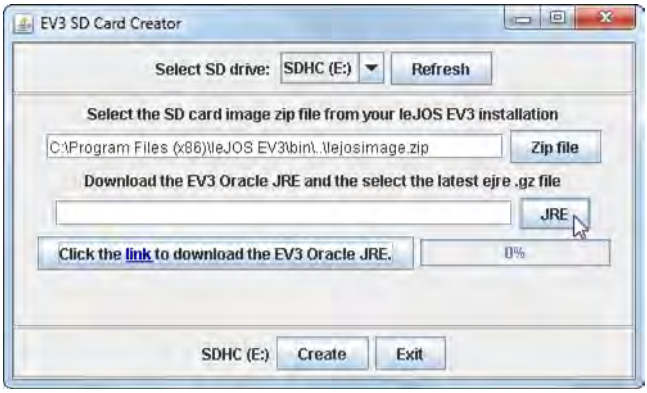
Product / File Description	File Size	Download
ARMv5 Linux - Headless EABI, SoftFP ABI, Little Endian[1]	32 MB	ejre-7u60-b19-ejre-7u60-fcs-b19-linux-arm-sflt-headless-07_may_2014.tar.gz

Nun ist der Download-Link »ejre-7u60…« unter »Oracle Java SE Embedded version 7 Update 60« auszuwählen. Danach gelangt man zur Login-Seite von Oracle. Der Download kann nicht gestartet werden, bevor man sich mit einem gültigen Account eingeloggt hat. Sollte keiner zur Verfügung stehen, muss ein neuer Account erstellt werden. Dazu ist wie folgt vorzugehen:

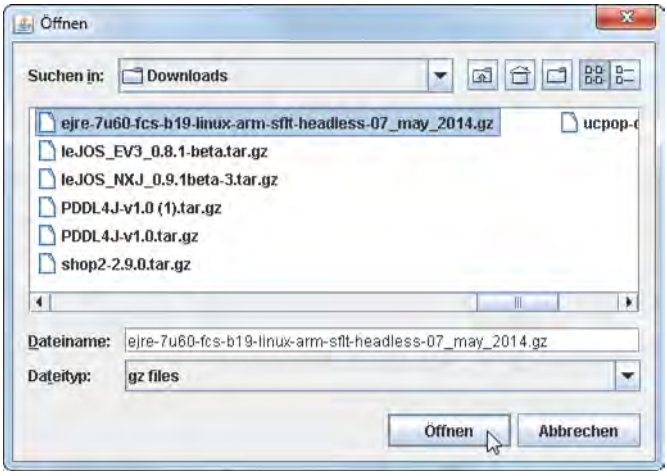
- Durch Klicken auf Account erstellen gelangt man zum Anmeldeformular.
- Alle Felder müssen ausgefüllt werden. Wichtig ist, dass die E-Mail-Adresse keine Fehler enthält, da Oracle eine Aktivierungs-Mail versendet.
- Wenn die Angaben vollständig sind und jedes Eingabefeld einen grünen Haken bekommen hat, geht es mit Account erstellen weiter.
- Der Bestätigungshinweis kann mit weiter bestätigt werden, woraufhin man wieder zur Login-Seite gelangt.
- Oracle versendet kurz nach der Anmeldung eine Bestätigungs-

mail. Sie enthält den Link »E-Mail-Adresse verifizieren«, der angeklickt werden muss, bevor der Account verwendet werden kann.

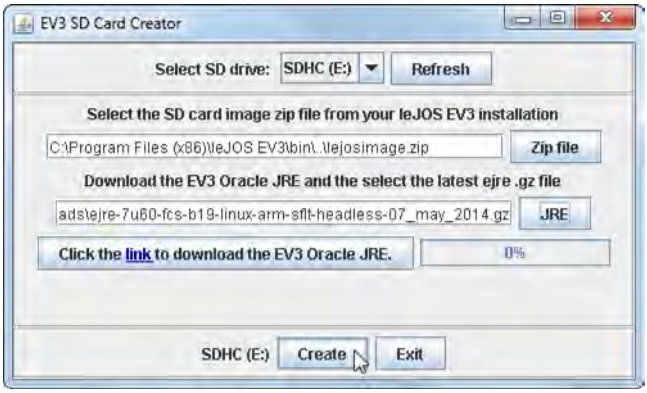
In der noch geöffneten Login-Seite kann der Download nun, nach Angabe des Benutzernamens (bei der Anmeldung verwendete E-Mail-Adresse) und des Passworts, durch einen Klick auf Anmelden gestartet werden. Nachdem dieser abgeschlossen ist geht es zurück zum »EV3 SD Card Creator«



Ein Klick auf JRE öffnet den Dateiexplorer.



Die eben heruntergeladene Datei ist nun auszuwählen und per Öffnen zu laden.



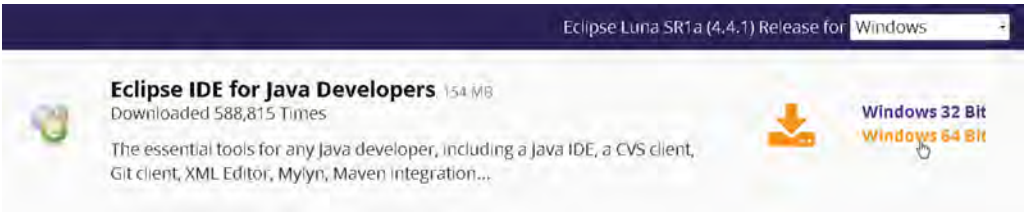
Mit Klick auf Create wird die microSD-Karte erstellt. Wenn sie fertig ist, erscheint eine Erfolgsmeldung, die mit OK bestätigt werden muss. Das Programm kann anschließend mit Exit geschlossen werden. Wichtig ist die microSD-Karte sicher zu entfernen, bevor sie vom Computer getrennt wird.

Die leJOS Firmware liegt jetzt in komprimierter Form auf der microSD-Karte. Sie kann nun in den EV3 Stein gesteckt und dieser durch Drücken der mittleren dunkelgrauen Taste (ENTER) gestartet werden. Beim ersten Start wird das komplette Linux-System auf der microSD-Karte aufgebaut. Dieser Vorgang kann bis zu 10 Minuten dauern.

Es könnte sein, dass der Start des neuen Systems wiederholt werden muss, weil es hängen geblieben ist. Wiederholt sich dieser Fehler, hilft nur die microSD-Karte noch einmal zu formatieren neu zu erstellen.

A.3 Installation Eclipse und leJOS Plug-in

Java Programme können mit jedem beliebigen Texteditor erstellt werden. Viel komfortabler ist jedoch die Verwendung einer Entwicklungsumgebung wie Eclipse. Unter http://www.eclipse.org/downloads/ kann diese heruntergeladen werden. Wir empfehlen die »Eclipse IDE for Java Developers« auszuwählen, denn darin ist alles enthalten was für die Programmierung des LEGO MINDSTORMS EV3-Roboters benötigt wird.



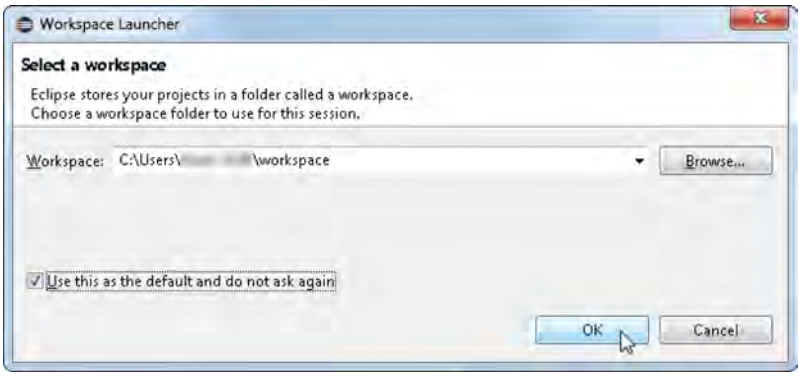
Hier ist, je nach System, die 32- oder 64-Bit Version auszuwählen. Danach gelangt man zur Auswahl des Download-Servers.

Der Download wird mit einem Klick auf den grünen Pfeil gestartet. Eclipse braucht nicht installiert zu werden, ein einfaches Entpacken der eben heruntergeladenen Datei reicht schon aus. Mit einem Rechtsklick darauf kann dies mittels »Alles extrahieren …« durchgeführt werden.

Wähle einen geeigneten Ort für Eclipse aus. Der Download Ordner ist nicht so gut geeignet, am einfachsten wäre `C:\`, aber auch `C:\Program Files\` (bei einer 64-Bit Version) oder `C:\Program Files (x86)\` (bei einer 32-Bit Version) sind gute Alternativen.

Nach dem Entpacken wird direkt das richtige Verzeichnis angezeigt. Per Rechtsklick auf »eclipse.exe« → »Senden an« → »Destop (Verknüpfung erstellen)« wird eine Verknüpfung auf dem Desktop erstellt, damit Eclipse zukünftig bequem von dort geöffnet werden kann.

Wird Eclipse nun gestartet, erscheint zuerst die Frage nach dem sogenannten »workspace«. Damit ist das Verzeichnis gemeint, in dem alle Eclipse-Projekte (und damit auch die Java-Klassendateien) abgespeichert werden.

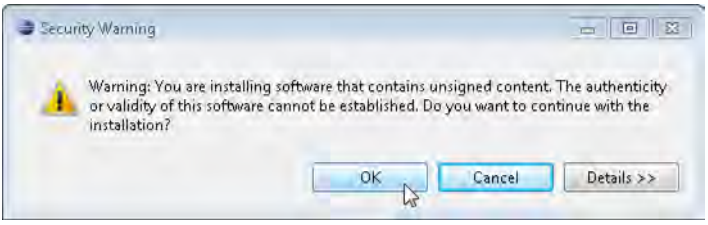


Der vorgeschlagenen Ordner muss nicht verändert werden. Um diese Abfrage nicht bei jedem Start angezeigt zu bekommen, kann das Häkchen bei »Use this as the default …« gesetzt werden. Nach einem Klick auf OK wird die Eclipse Entwicklungsumgebung vollständig geladen. Das Begrüßungsfenster kann mit einem Klick auf das Kreuz oben links im Tab »Welcome« geschlossen werden schließen.

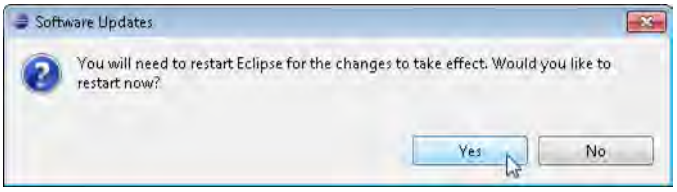
Bevor es richtig losgeht muss noch das leJOS EV3 Plug-in installiert werden. Am einfachsten funktiert dies über den sogenannten »Eclipse Marketplace«, der über »Help« → »Eclipse Marketplace…« aufgerufen werden kann.

Im Suchfeld kann nun einfach »lejos« eingetragen und per Klick auf die kleine Lupe nach den entsprechenden Plug-ins gesucht werden

Es werden nun alle verfügbaren leJOS Plug-ins (für den NXT und den EV3) angezeigt. Per Klick auf Install neben dem Eintrag für den EV3 wird dieser ausgewählt. Im nun erscheinenden Fenster muss nach einem Klick auf Confirm > noch die Lizenzvereinbarung akzeptiert werden. Der Button Finish startet jetzt die Installation.



Während der Installation erscheint die Sicherheitswarnung, dass diese Software unsigniert ist. Diese wird einfach mittels OK bestätigt, da wir der leJOS Community vertrauen.



Eclipse muss nun neugestartet werden, damit das EV3 Plug-in in der Entwicklungsumgebung zur Verfügung steht. Die Nachfrage wird also mit Yes beantwortet.

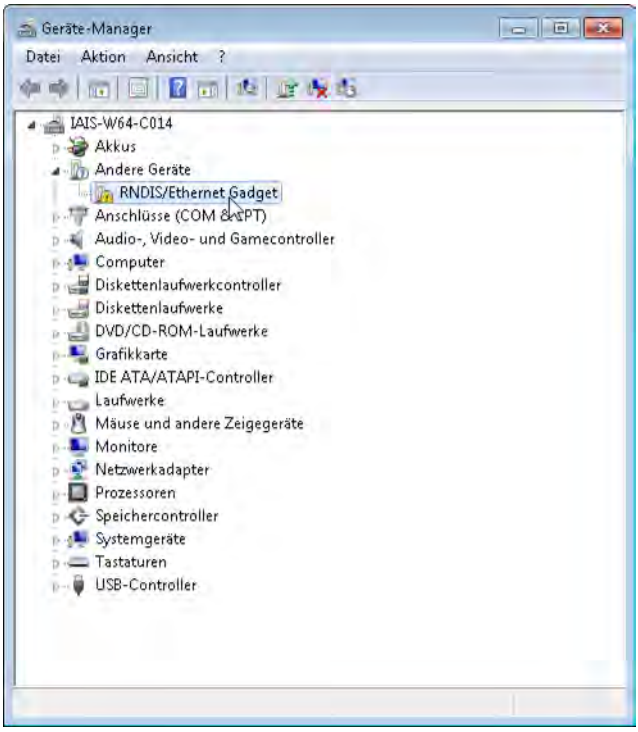
Im neu gestarteten Eclipse-Fenster ist nun in der Menüleiste der Eintrag »leJOS EV3« zu sehen. Damit kann das schon bekannte Programm »EV3 SD Card Creator« und das »EV3 Control Center« gestartet werden.

A.4 EV3 über USB verbinden

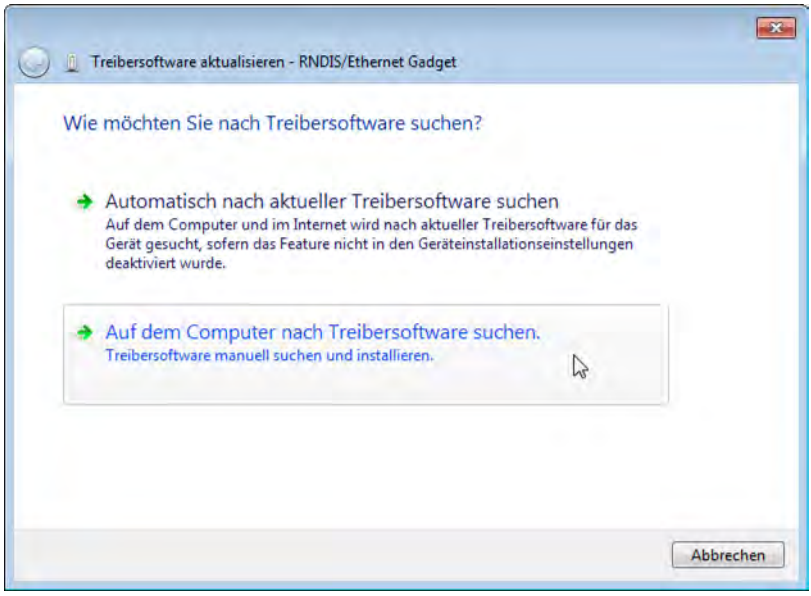
Im Folgenden wird beschrieben, wie die Verbindung zwischen EV3 und Computer per USB aufgebaut werden kann. Dies ist im Gegensatz zur Verbindung über WLAN sehr simpel. Sobald die nötigen Treiber einmal installiert sind wird die Verbindung automatisch durch Einstecken des USB-Kabels am Computer und am EV3 hergestellt. Diese Methode hat aber den Nachteil, dass ein EV3-Roboter immer wieder vom Kabel getrennt werden muss bevor er zum Beispiel Bewegungen im Raum ausführen kann.

Um also den Treiber zu installieren muss zunächst der EV3 mittels USB-Kabel mit dem Computer verbunden werden. Windows signalisiert durch ein Geräusch, dass neue Hardware mit dem Computer verbunden wurde und versucht automatisch die richtigen Treiber zu finden. Dies wird auch in der Startleiste durch ein kleins Icon angezeigt, bleibt aber erfolglos.

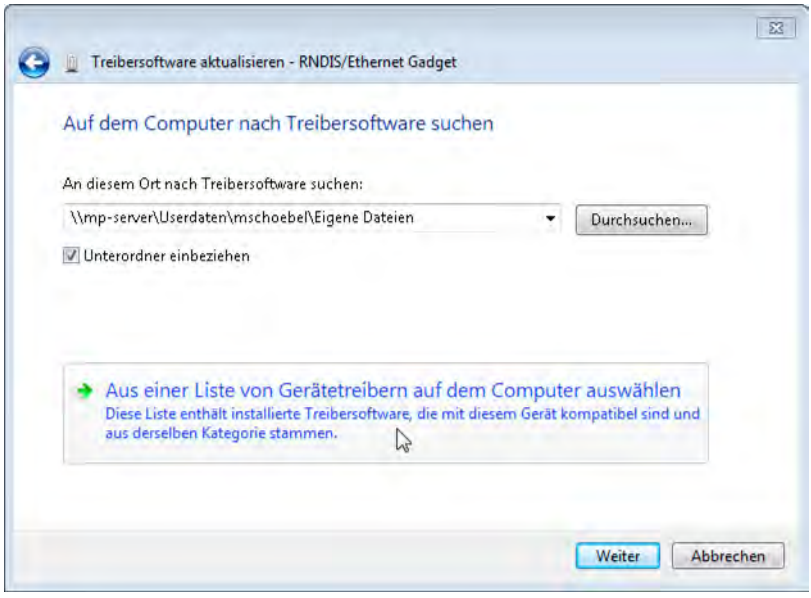
Durch Klick auf »Start« → »Systemsteuerung« → »Hardware und Sound« → »Geräte-Manager« gelangt man zum Geräte-Manager, der für die Verwaltung der Hardware zuständig ist.



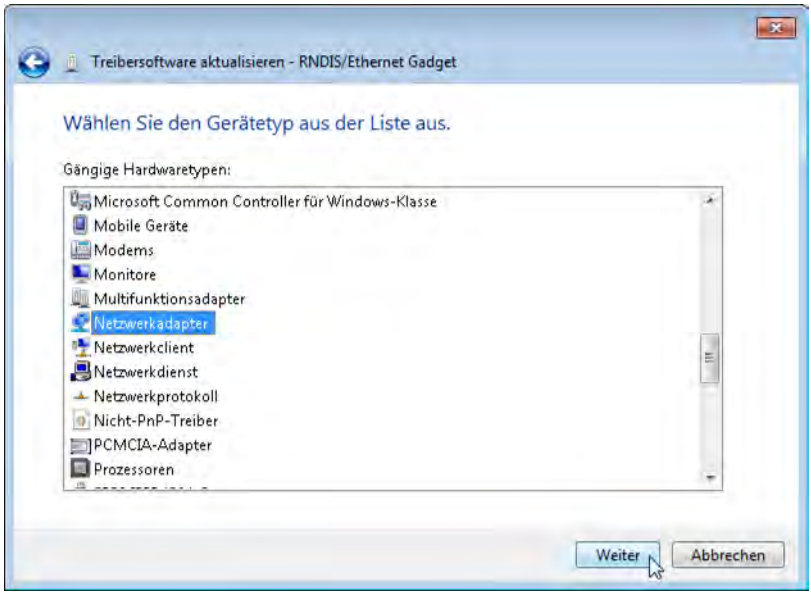
Unter »Andere Geräte« wird ein nicht erkanntes Gerät durch ein kleines Ausrufezeichen angezeigt. Dies ist die Schnittstelle zum EV3. Nach einem Rechtsklick darauf und der Auswahl von »Treibersoftware Aktualisieren...« gelangt man zum Treiber-Installationsassistenten.



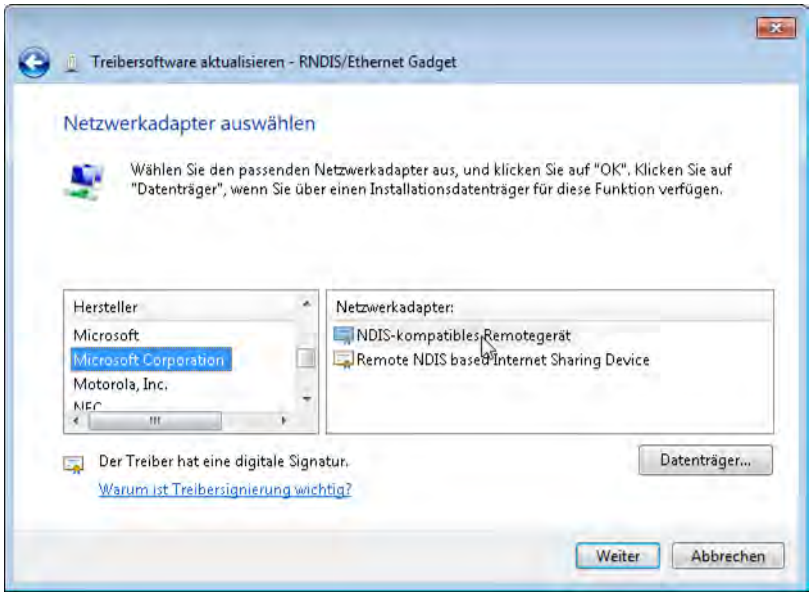
Hier ist »Auf dem Computer nach Treibersoftware suchen.« auszuwählen, da die automatische Suche nicht zum Ziel führt.



Nach einen Klick auf die angezeigte Schaltfläche »Aus einer Liste von …«, gelangt man zur Auswahl des Gerätetyps.

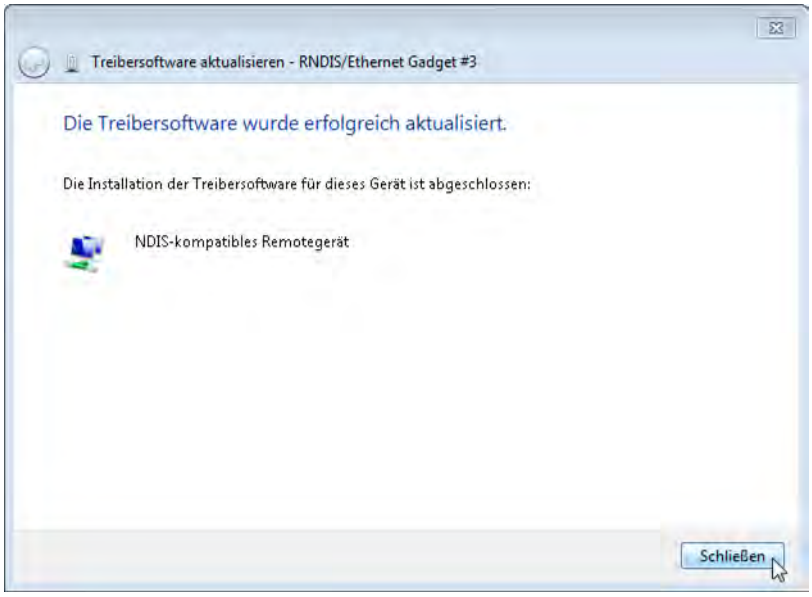


In der Liste ist »Netzwerkadapter« auszuwählen. Durch Klick auf weiter gelangt man zur Treiberauswahl nach Hersteller und Modell.



Unter »Microsoft Corporation« als Hersteller ist ein »NDIS-kompatibles Remotegerät« aufgeführt. Dies wird markiert und mit Weiter bestätigt.

Ein Klick auf Ja bei der angezeigten Warnung startet die Installation.



Herzlichen Glückwunsch. Die Treiberinstallation wurde erfolgreich abgeschlossen. Nachdem das Installationsfenster mittels Schließen geschlossen wurde, wird die Schnittstelle zum EV3 im Geräte-Manager als »RNDIS-Etherner Gadget« (ohne kleines Ausrufezeichen) angezeigt. Die Verbindung über USB ist damit einsatzbereit.

Falls es zukünftig Probleme mit der Verbindung geben sollte, werden in Kapitel 9 ab Seite 141 einige Möglichkeiten angegeben um dise zu beheben.

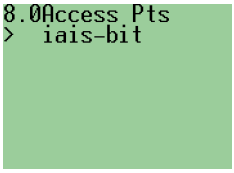
A.5 EV3 über WLAN verbinden

Die Verbindung des EV3 über WLAN benötigt ein bestehendes WLAN-Netzwerk, ist damit in der weiteren Benutzung aber komfortabler, da nicht kabelgebunden. Der EV3 kann somit beim Überspielen eines neuen Programms über Eclipse einfach an seiner Position im Raum verbleiben.

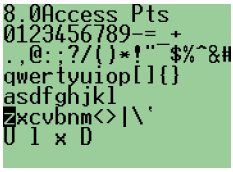
Damit eine WLAN-Verbindung mit dem EV3 hergestellt werden kann, müssen sich sowohl der Computer, als auch der EV3 im selben WPA2 verschlüsselten WLAN-Netzwerk befinden. Wir nehmen an, dass der Computer mit dem entsprechenden Access Point verbunden ist, sich also schon in dem Netzwerk befindet. Im Folgenden wird beschrieben, wie der EV3 mit dem Access Point verbunden wird.

Dazu muss der EV3 zunächst mit eingestecktem WLAN-USB-Adapter gestartet werden (in dieser Anleitung wird der »Netgear N150 (WNA1100)« USB-Adapter verwendet). Im Hauptmenü muss dann zum Menüpunkt »Wifi« navigiert und dieser per ENTER-Taste ausgewählt werden (vgl. Abschnitt 6.3 auf Seite 76)

Daraufhin werden alle in der Umgebung gefundenen Access Points angezeigt. Mit den UP- und DOWN-Tasten muss derjenige ausgewählt werden mit dem auch der Computer verbunden ist. Die Auswahl ist mit ENTER zu bestätigen.



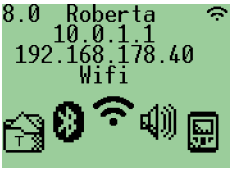
Die Verbindung benötigt die Eingabe des Passworts. Mit Hilfe der UP, DOWN, LEFT und RIGHT Tasten des EV3 können die verschiedenen Zeichen markiert werden. Per ENTER wird das aktuell angewählte Zeichen für das Passwort ausgewählt und erscheint in der letzten Zeile. Die zweitletzte Zeile »U l x D« ist dabei die Menüsteuerung für den Eingabebildschirm:



- **U** steht für »UPPER« und aktiviert die Eingabe von Großbuchstaben (ähnlich der Hochstelltaste auf der Computertastatur)
- **l** steht für »lower« und aktiviert die Eingabe von Kleinbuchstaben
- **x** steht für »delete« und löscht das zuletzt eingegebene Zeichen
- **D** steht für »Done« und schickt die bisher eingegebenen Zeichen

(die nun in der letzten Zeile zu sehen sind) als Passwort an das System

Nachdem das Passwort komplett eingegeben und mittel **D** bestätigt wurde, verbindet sich der EV3 mit dem Netzwerk. Daraufhin kehrt die Anzeige zum Hauptmenü zurück.



Es hat geklappt. Der EV3 hat vom Access Point eine IP-Adresse zugewiesen bekommen (hier: `192.168.178.40`). Diese kann in der dritten Zeile des Hauptmenüs abgelesen werden. Ausserdem erscheint rechts oben in der Ecke das kleine Wifi-Symbol. Die Verbindung zwischen EV3 und Computer über WLAN ist nun einsatzbereit.

Ein neues Projekt Anlegen

In diesem Abschnitt wird anhand eines Beispiels Schritt für Schritt erklärt, wie in Eclipse ein neues Projekt und darin eine neue Klassendatei angelegt wird. Wenn das erledigt ist, kann es direkt mit der Programmierung des ersten Programms, wie in Abschnitt 6.1 auf Seite 73 beschrieben, losgehen.

In Eclipse wird über File → New → Project ein neues Projekt angelegt (siehe Abbildung B.1).

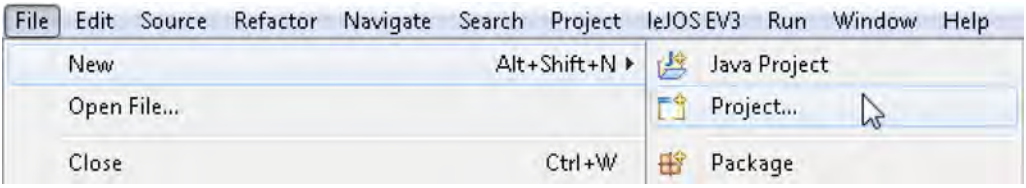


Abbildung B.1.: Anlegen eines neuen Projektes in Eclipse

Im darauf folgenden Fenster wird »LeJOS EV3 Project« ausgewählt und mit Next bestätigt (siehe Abbildung B.2 auf der nächsten Seite).

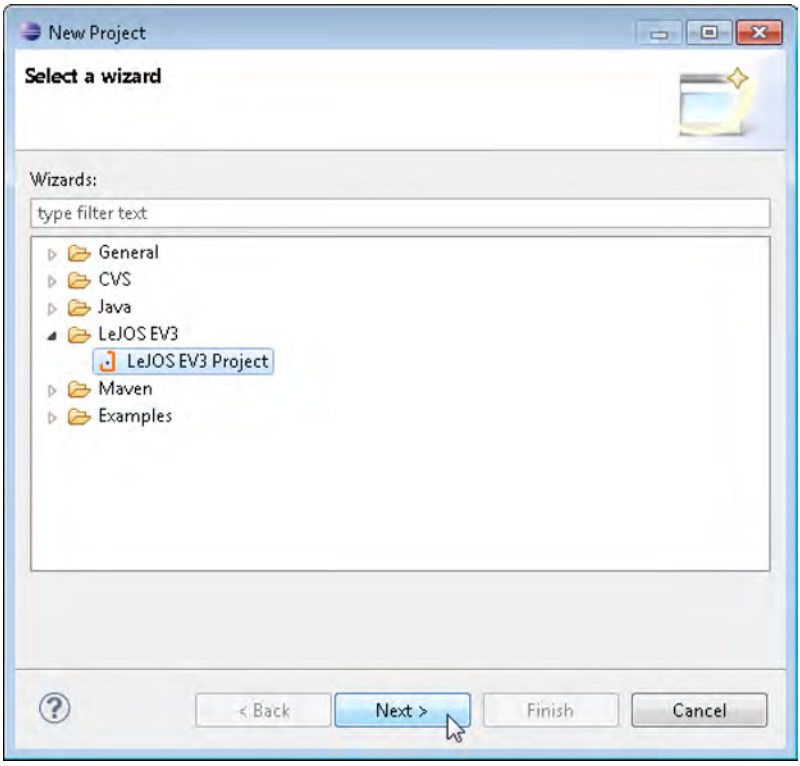


Abbildung B.2.: leJOS EV3 Project erstellen

Nun muss ein Name für das Projekt festgelegt werden. Der Projektname ist beliebig, darf aber nicht schon im »Package Explorer« vorhanden sein. Im dargestellten Beispiel wurde der Name »EV3FirstProgram« gewählt (vgl. Abbildung B.3 auf der nächsten Seite).

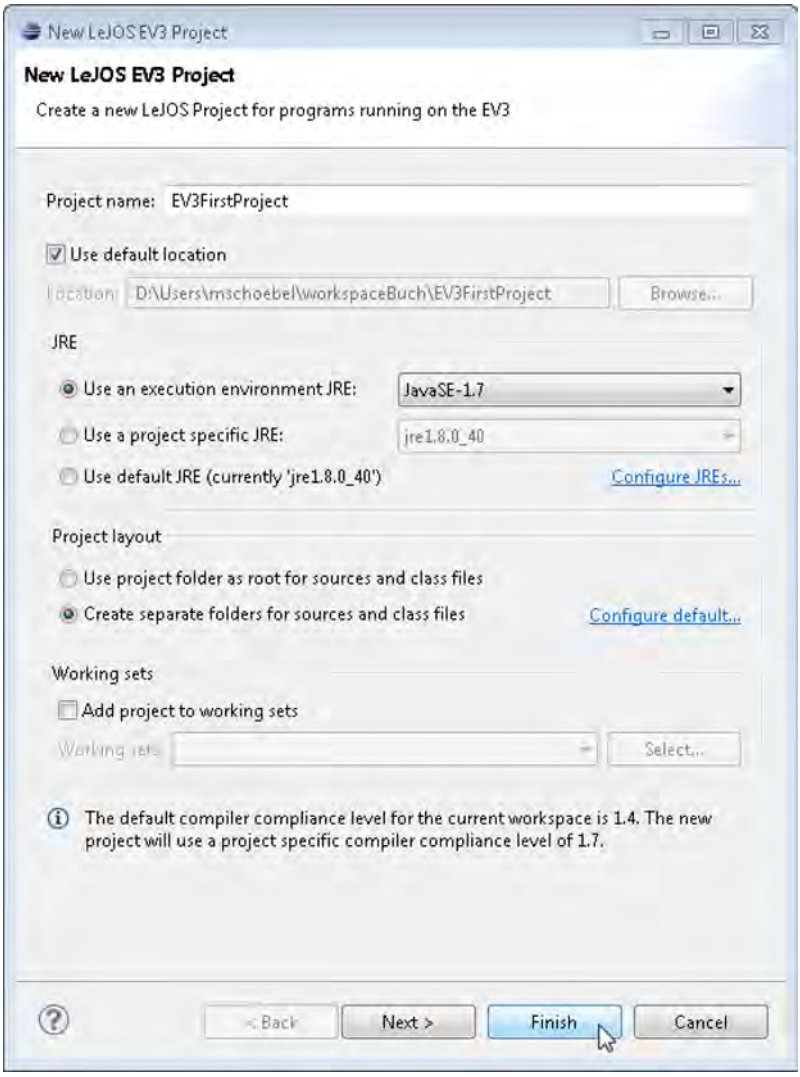


Abbildung B.3.: Anlegen des Java-Projektes EV3FirstProgram in Eclipse

Mit Klick auf die Schaltfläche Finish wird das neue Projekt angelegt und befindet sich nun mit den anderen schon vorhandenen Projekte in Eclipse links im »Package Explorer«.

Eine neue Klasse anlegen

Nun wird im neuen Projekt eine Klasse angelegt. Dies geschieht mit Doppelklick auf das erstellte Projekt »EV3FirstProject« und anschlie-

ßend Rechtsklick auf src → New → Class, wie in Abbildung B.4 dargestellt.

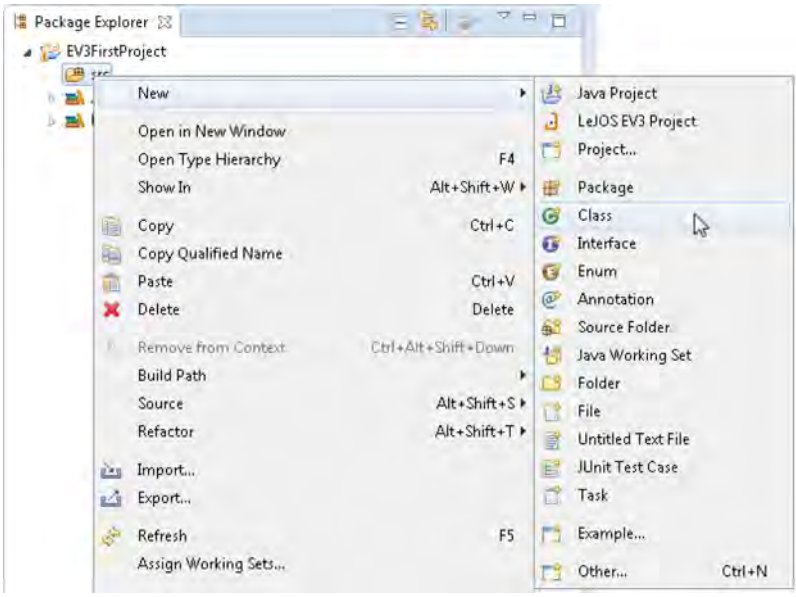


Abbildung B.4.: Anlegen einer neuen Klasse im Projekt »EV3FirstProgram«

Im darauf folgenden Fenster können Einstellungen zur neuen Klasse vorgenommen werden. Wichtig ist, dass diese Klasse einen Namen erhält, also beispielsweise »HelloWorld«. Durch Setzen des Häkchens bei »public static void main(String[] args)« wird die `main`-Methode automatisch mitgeneriert (vgl. Abbildung B.5 auf der nächsten Seite). Diese könnte auch später manuell eingefügt werden. In jedem Programm wird eine solche `main`-Methode benötigt, da diese als Hauptmethode erkannt und das Programm dort gestartet wird.

Abbildung B.5.: Einstellungen der neuen Klasse im Projekt »EV3FirstProgram«

Mit Finish wird die neue Klasse angelegt und damit der folgende Code generiert:

Programm B.1: Generierte HelloWorld-Klasse

```
1 public class HelloWorld {
2
3     public static void main(String[] args) {
4         // TODO Auto-generated method stub
5
6     }
7 }
```

Geschichte zu Java

Die Anfänge von Java gehen auf das Projekt »Green« zurück, das Sun Microsystems im Dezember 1990 als Ergebnis einer internen Diskussion über die Zukunft der IT-Industrie startete. Eine Gruppe talentierter Programmierer und Visionäre – zum »harten Kern« der Innovatoren gehörten James Gosling, Patrick Naughton und Mike Sheridan – machten sich daran, »die Welt zu verändern«. Sie mieteten – abseits des Firmengeländes von Sun Microsystems – ein Büro in der Sand Hill Road in Menlo Park an und gingen zum Einkaufen – Geräte, wie sie in jedem Haushalt zu finden waren. Die Forscher zerlegten und studierten diese Geräte, um daraus Technologietrends für die Zukunft abzuleiten, und um einige der zerlegten Teile als Komponenten für ein neues Gerät wiederzuverwenden, das als Prototyp entwickelt werden sollte und später den Namen *7 (sprich «star seven«) bekam. Der Name für den *7 leitete sich übrigens auch von einem Gerät ab, und zwar der dortigen Telefonanlage. Wenn ein Anruf kam, und keiner ans klingelnde Telefon ging, konnte man durch die Eingabe von *7 den Anruf zu sich auf den eigenen Telefonapparat holen. Da zunächst keiner mit dem Code *7 vertraut war, musste diese Zeichenkombination öfter durchs Büro gerufen werden, bis sie sich jeder eingeprägt hatte. Später rief man einfach laut *7, wenn man nicht ans Telefon gehen wollte, und schließlich war es der Bürowitz schlechthin, den neuen Geräteprototypen so zu benennen. Die Ingenieure waren überzeugt, dass das nächste Zeitalter des Computing durch kleine Geräte bestimmt würde, und dass eine Fokussierung auf haushaltsübliche Geräte, wie zum Beispiel Videorekorder oder Telefone, stattfinden würde. Der *7 sollte einen Prototyp für die nächste Welle konvergierender Technologien zwischen Geräten sein, wie sie in jedem Haushalt zu finden waren und Geräten der Informationstechnologie darstellen. Das Projektteam »Green« untersuchte zuerst Haushaltsgeräte und dabei erkannte man, dass Mikroprozessoren in allen gängigen elektrischen Haushaltsgeräten integriert werden – vom Telefon bis zum Toaster.

Mit dem *7 sollte darum der Prototyp eines Haushaltsgerätecontrol-

lers entwickelt werden. Er war ein kleiner Computer, mit einem 5 Zoll großen und 16 Bit tiefen, farbigen LCD Display, einer SPARC CPU, kleinen Lautsprechern und diversen Schnittstellen(vgl. Abbildung C.1 auf der nächsten Seite). Man portierte auf ihn eine Version des Betriebssystems UNIX, die mit weniger als 1 MByte Hauptspeicher auskam und implementierte ein Dateisystem auf seinem Flashspeicher. Als Programmiersprache sollte C++ eingesetzt werden. Signifikante Probleme mit den Werkzeugen, eine hohe Fehlerrate, schwierige bis fehlende Portabilität, Sicherheitslücken, etc. führten jedoch dazu, dass James Gosling, der heute als Vater von Java gilt, für einen alternativen Ansatz plädierte und begann, eine neue Sprache für den *7 zu entwickeln. Die wichtigsten Anforderungen für die neue Sprache waren:

- Netzwerkunterstützung: Es müssen Daten und Programme zwischen verschiedenen Geräten ausgetauscht werden können.
- Sicherheit: Es dürfen keine unerlaubten Zugriffe auf sensitive oder schutzwürdige Informationen möglich sein.
- Stabilität: Verbrauchergeräte sollen weder »abstürzen« noch gelegentlich neu booten.
- Platformunabhängigkeit: Es soll eine Kompatibilität zwischen einer Vielzahl von Geräten ermöglicht werden.
- Mehrläufigkeit: Es müssen mehrere Aktionen/Programme gleichzeitig ausgeführt werden können.
- Dynamik: Das Laden und Entfernen von Programmen muss auch ohne lokale Festplatte möglich sein.
- Kleine Programmcodegröße: Kleiner Programmcode ermöglicht preiswerte Geräte mit kleinem Speicher.
- Einfach und vertraut: Die Sprache soll das Leben für den Programmierer leichter machen, durch Ähnlichkeiten zu C und C++.

Mit den Anforderungen an den *7 wurden quasi schon die Anforderungen an die Programmiersprache Java definiert, die zunächst unter dem internen Codenamen OAK (Object Application Kernel) entwickelt wurde. Die Ingenieure zerlegten abermals eine Reihe von Produkten und Haushaltsgeräten und bauten sie so wieder zusammen, dass sie miteinander kommunizieren, Objekte an die jeweils anderen weitergeben und ihr Verhalten gegenseitig verstehen konnten. Ziel war es, das Zusammenspiel der Geräte zu verbessern. Dadurch ließe sich ihre Herstellung vereinfachen und Verbraucher könnten sie ohne weiteres miteinander verbinden.

Am Ende stellte der *7 verschiedene Anwendungen bereit, so z. B. eine Fernbedienung, einen elektronischen TV-Programmführer, eine Scheckbuchanwendung und einen Termin-Kalender. Bei Verwendung als Fernbedienung, wurde die passende Benutzerführung (für den Fernseher oder den Videorekorder) zur Laufzeit nachgeladen.

Der Benutzer agierte mit dem *7 über einen grafischen Assistenten, der heute Duke heißt und inzwischen das Maskottchen von Java geworden ist.

Abbildung C.1.: Star 7

Duke führte den Anwender durch den virtuellen Haushalt, half bei der Programmierung des Videorekorders, las Magazine und navigierte von Zimmer zu Zimmer. Der Prototyp, und insbesondere die ihr zugrunde liegende Technologie, wurde von Sun Microsystems begeistert aufgenommen. Eine eigene Firma namens FirstPerson wurde ins Leben gerufen. Ziel des Unternehmens war, OAK den wichtigen Herstellern von Consumerelektronik-Geräten nahe zubringen.

FirstPerson startete Verhandlungen mit zahlreichen Firmen und bewarb sich u. a. um die Entwicklung von Set-Top Boxen – denen man bereits

damals die Zukunft des interaktiven TV zusprach. Doch alle Verhandlungen scheiterten: Der Markt war offenbar noch nicht reif für das, was FirstPerson anzubieten hatte. 1994 kam dann das Aus für FirstPerson. Niemand hatte OAK lizenzieren wollen. Die Firma wurde aufgelöst. Zu der Zeit war die Welt bereits vom Internet-Fieber erfasst, und das Web hatte eine drastische Wende erfahren – weg vom ausschließlichen Medium für Forschung und Wissenschaft, hin zu einem Consumer-Vehikel als neues Medium für die IT.

Sun Microsystems erkannte den Bedarf an einer sicheren, plattformunabhängigen Programmierumgebung für das Internet und orientierte sich mit OAK nun in ebendiese Nische. Die Sprache wurde um zusätzliche Sicherheitskomponenten erweitert und zum Download ins Internet gestellt. Web-Anwender und -Anbieter nahmen die neue Sprache, die nun den Namen Java trug, begeistert auf. Sun Microsystems führte mit ihr seinen traditionell »offenen« Ansatz fort und stellte Compiler, Spezifikationen, Bibliotheken und Dokumentation frei zugänglich in das Internet. Java bot die einzigartige Möglichkeit, Anwendungen erstellen und verteilen zu können, die sich für jedes Netzwerk und jedes Betriebssystem eignen. Heute ist die Sprache allgegenwärtig und in unterschiedlichsten Geräten wie Chipkarten, Set-Top-Boxen, mobilen Telefonen, Laptops, PCs und Großrechner im Einsatz. Weltweit gibt es mehr als sechs Milliarden Geräte, die auf Java basieren. Gerade für große Unternehmen ist Java durch seine Plattformunabhängigkeit und Sicherheit von großer Bedeutung.

Java ist heute nicht mehr nur eine Programmiersprache, sondern ein ganzes Ökosystem: Es gibt weltweit eine riesige Community aus Entwicklern und Benutzern, die Java-Technologien nutzen und vorantreiben. Die Innovationen im Java-Umfeld wurden durch diverse Open-Source Projekte in den vergangenen Jahren und durch Veröffentlichung des Java-Quellcodes extrem beschleunigt. Folgende Liste zeigt einige Beispiele für den Einsatz von Java in verschiedenen Bereichen:

- Im Handel, wie etwa in Kaufhäusern: Java-Applikationen steuern die weltweiten Warenbewegungen und helfen dabei, neue Strategien zu entwickeln.
- Produzierendes Gewerbe, beispielsweise Automobilhersteller: Hier ist Java im Einsatz zur Prozessüberwachung und -steuerung.
- Banken: Java spielt eine wichtige Rolle beim Online-Banking, in EC-Automaten sowie im weltweiten Netz von Finanztransaktionen aller Art.
- Festnetz und Mobilfunkbetreiber: Java Applikationen finden sich

auf Handys sowie bei der Überwachung und Steuerung der Funk- und Festnetze.

- Realtime Java mit garantierten Antwortzeiten wird zum Beispiel im Einsatz bei Flugsimulatoren benutzt.
- Java wird im Zusammenspiel mit der Funkfrequenztechnik RFID eingesetzt.
- Embedded Java findet sich in Mikrocontrollern, Routern, Appliances, etc.
- Java ist eine der Basistechnologien des Internets, zum Beispiel bei Buchungssystemen (z.B. Reisen), Marktplätzen (z.B. eBay), Auskunftsystemen (Bahn, Telefonbuch) etc.
- Java ist die Basis vieler IT-Anwendungslandschaften in kleinen und großen Unternehmen

Literatur

Bredenfeld, Ansgar und Thorsten Leimbach (2010). „The Roberta® Initiative". In: Proceedings of the 2nd International Conference on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS (SIMPAR 2010). Darmstadt, S. 1–2 (siehe S. 3).

Brooks, Rodney A. (1985). *A robust layered control system for a mobile robot*. Techn. Ber. Massachusetts Institute of Technology, Artificial Intelligence Laboratory. URL: http://dspace.mit.edu/bitstream/handle/1721.1/6432/AIM-864.pdf (siehe S. 126, 127).

Bundesagentur für Arbeit, Hrsg. (2014). *Verteilung der Arbeitslosigkeit*. URL: http://statistik.arbeitsagentur.de/Statischer-Content/Arbeitsmarktberichte/Akademiker/generische-Publikationen/Broschuere-Akademiker-2012.pdf (besucht am 02. 06. 2014) (siehe S. 2).

DESTATIS, Hrsg. (2014). *Bildung und Kultur*. URL: https://www.destatis.de/DE/Publikationen/Thematisch/BildungForschungKultur/Hochschulen/PruefungenHochschulen2110420127004.pdf?__blob=publicationFile (besucht am 02. 06. 2014) (siehe S. 3).

Deutsches Institut für Wirtschaftsforschung, Hrsg. (2014). *Ingenieurmangel*. URL: https://www.diw.de (besucht am 02. 06. 2014) (siehe S. 2).

Druin, Allison und James A. Hendler, Hrsg. (2000). *Robots for Kids. Exploring New Technologies for Learning*. San Francisco: Morgan Kaufmann (siehe S. 4).

Frank, Elisabeth (1995a). „Anregungen für den Physikunterricht. Physik - ein Fach für Mädchen und Jungen". In: Hrsg. von Weiterbildung und Kunst Baden Württemberg für Familie Frauen. Schule der Gleichberechtigung. Stuttgart, S. 111–127 (siehe S. 6).

Frank, Elisabeth (1995b). „Mädchen und Physik - wider die Natur?" In: Hrsg. von Helga Krahn und Cornelia Niederdrenk-Felgner. Frauen machen Schule. Bielefeld: Kleine Verlag. Kap. hallo, S. 112–125 (siehe S. 6).

Häußler, Peter und Lore Hoffmann (1998). *Erläuterungen zu Modul 7 mit Unterrichtsbeispielen für den Physikunterricht*. Techn. Ber. Kiel: BLK-Programmförderung »Steigerung der Effizienz des

mathematisch-naturwissenschaftlichen Unterrichts«. Institut für die Pädagogik der Naturwissenschaften an der Universität Kiel (siehe S. 6).

ITWissen. *Java Package*. 2014-08-05. URL: http://www.itwissen.info/definition/lexikon/Java-Package.html (siehe S. 56).

Kayser, Ina und Oliver Koppel (2013). *Ingenieurmonitor*. Techn. Ber. Köln: Verein Deutscher Ingenieure (siehe S. 2).

Kompetenzzentrum Technik - Diversity - Chancengleichheit e.V., Hrsg. (2014). *Daten + Fakten Studium*. URL: http://www.kompetenzz.de/Daten-Fakten/Studium (besucht am 02. 06. 2014) (siehe S. 3).

Krüger, G. and Hansen, H. (2014). *Java-Programmierung - Das Handbuch zu Java 8*. 7. Auflage von 2011 unter http://www.javabuch.de/download.html kostenlos herunterladbar. Köln: O'Reilly Verlag (siehe S. 64).

LEGO Education, Hrsg. (2013). *LEGO MINDSTORMS Education EV3 Press Kit*. URL: http://educationnews.legoeducation.us/Press-Kits/132/LEGO-MINDSTORMS-Education-EV3-Press-Kit (besucht am 28. 11. 2013) (siehe S. 91, 97).

Oracle, Hrsg. (2014). *Technical Roadmap Endorsed by All Major Java Players*. URL: http://www.oracle.com/us/corporate/press/193190 (besucht am 05. 10. 2014) (siehe S. 19).

Papert, Seymor (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books (siehe S. 4).

Ullenboom, Christian (2011). *Java ist auch eine Insel - Das umfassende Handbuch*. 10. Auflage von 2011 unter http://openbook.rheinwerk-verlag.de/javainsel/ kostenlos herunterladbar. Rheinwerk Verlag (siehe S. 27).

Wikipedia (2014). *Unified Modeling Language*. URL: http://de.wikipedia.org/wiki/UML (besucht am 05. 08. 2014) (siehe S. 54).

Abbildungsverzeichnis

Programmverzeichnis

Stichwortverzeichnis

Vorstellung weiterer Bücher zu Java

Roberta – Programmieren mit Java **Leimbach, Bredenfeld**

Inhalt Band 3 - NXT der Roberta-Reihe ergänzt die in Band 1 – NXT vorgestellten Programmiermöglichkeiten (mit NXT-G und NXC) um Java (leJOS). Dabei werden neben den Grundlagen der objektorientierten Programmierung insbesondere auch auf die Umsetzung von Java auf das LEGO MINDSTORMS NXT System mit leJOS behandelt. Die Beschreibung reicht von der Installation von leJOS bis zur Umsetzung größerer Experimente unter leJOS. Dieser Band richtet sich an Personen die bereits erste Erfahrung mit textuellen Programmiersprachen haben.

2010, 165 S., zahlr. Abb. u. Tab., Kartoniert
Sprache: Deutsch
Fraunhofer Verlag
ISBN: 978-3-8167-8401-2

Java ist auch eine Insel: Insel 1: Das umfassende Handbuch **Ullenboom**

Inhalt Java-Einsteiger, Studenten und Umsteiger profitieren seit mehr als einem Jahrzehnt von diesem Lehrwerk. Neben der Behandlung der Sprachgrundlagen von Java (Ganz neu: Lambda-Ausdrücke!) gibt es kompakte Einführungen in Spezialthemen. So erfahren Sie einiges über Threads, Swing, Netzwerkprogrammierung, NetBeans, RMI, XML und Java, Servlets und Java Server Pages. Dieses Buch gehört in das Regal eines jeden Java-Programmierers. Es liegt hiermit in der 11. Auflage vor.

2014, 1306 S., zahlr. Abb. u. Tab., Kartoniert
Sprache: Deutsch
Verlag: Galileo Computing ISBN: 978-3-8362-2873-2

Bagnall **Maximum Lego Ev3: Building Robots with Java Brains**

Inhalt The LEGO Mindstorms EV3 set is the latest in robotics technology, allowing you to build incredible motorized inventions without knowing anything about electronics. As a mass market device with hundreds of pieces and a suite of available programming tools, it can be both powerful and overwhelming. This book walks readers through an in depth introduction to the kit, laying down the fundamental principles of robotics and Java programming, while delivering dozens of spectacular robot projects. This knowledge will maximize your creativity and give your robots the abilities they need to handle almost any situation. Complete 3-D-rendered building instructions for eight robots are included,

2014, 464 S., zahlr. Abb. u. Tab., Kartoniert
Sprache: Englisch
Verlag: Variant Press
ISBN: 978-0-9868-3229-1

Berns, Schmidt **Programmierung mit LEGO Mindstorms NXT: Robotersysteme, Entwurfsmethodik, Algorithmen**

Inhalt Das Buch bietet anhand von LEGO Mindstorms NXT-Robotern einen anschaulichen Einstieg in die Informatik. Nach einer allgemeinen Einführung in informationstechnische Prinzipien und Komponenten werden in Teil 2 die NXT-Elemente behandelt. Dazu gehören die Bauteile und ihre Funktionsweisen sowie die grafische Programmierumgebung. Teil 3 führt in die objektorientierte Programmiersprache JAVA und die Programmierung von LEGO-Robotern ein.

2010, 248 S., zahlr. Abb. u. Tab., Kartoniert
Sprache: Deutsch
Verlag: Springer
ISBN: 978-3-6420-5469-3

Java-Programmierung - Das Handbuch zu Java 8

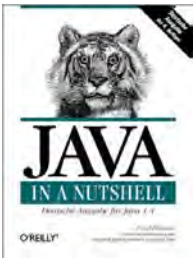
Krüger, Hansen

Inhalt Die Neuauflage dieses Standardwerks führt Sie umfassend in die Programmierung mit Java 8 ein. Vom Aufbau einer funktionierenden Entwicklungsumgebung über Grundlagen der Sprache bis hin zu Themen wie Grafik-, Netzwerk- oder Datenbankprogrammierung werden alle wichtigen Eigenschaften der Java 8 Standard Edition vorgestellt. Auch die aktuellen Schlüsselthemen wie funktionale Interfaces, Lambda-Ausdrücke, Closures und Methoden-Referenzen werden mit vielen Beispielen umfassend und leicht verständlich erläutert. Daneben runden eine Vielzahl weiterführender Themen das Werk ab, wie beispielsweise Design Patterns, Kryptografie oder XML.

2014, 1104 S., zahlr. Abb. u. Tab., Kartoniert,
Sprache: Deutsch Verlag: O'Reilly ISBN: 978-3-9556-1514-7

Java in a Nutshell

Flanagan, Dalheimer



Inhalt Obgleich es grundsätzlich als Referenzwerk konzipiert ist, beginnt das Buch mit einer ausführlichen, zügigen Einführung in Java, die sämtliche Schlüsselthemen wie Syntax, objektorientierte Programmierung, Sicherheit, Beans und Tools abdeckt. Die Erläuterungen zu den einzelnen Themen sind knapp gehalten und gleichzeitig von hohem Informationsgehalt, und wer sich dieses Buch mit dem Ziel kauft, die Sprache zu lernen, wird sich vom Tempo dieser Einführung vermutlich erst einmal überrumpelt fühlen. Dieses Buch erhebt den Anspruch, eine umfassende Dokumentation von Java zu sein. Es leistet diesem Anspruch Genüge durch die gelungene Darstellung der Java-Programmierung in Teil I und die paketbasierte API-Katalogisierung in Teil II.

2000, 748 S., Auflage: 3., zahlr. Abb. u. Tab., Kartoniert
Sprache: Deutsch
Verlag: O'Reilly
ISBN: 978-3-8972-1190-2