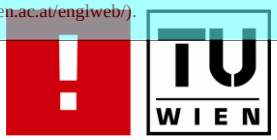
Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).



FAKULTÄT FÜR !NFORMATIK

Vermittlung objektorientierter Konzepte mittels LEGO Mindstorms

MAGISTERARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Dipl.-Ing. Bernhard Löwenstein
Matrikelnummer 9726426

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuerin: Ass.Prof. Dipl.-Ing. Dr. Monika Di Angelo

Wien, 13.11.2009 ....
(Unterschrift Verfasser) (Unterschrift Betreuerin)

Eidesstattliche Erklärung

Dipl.-Ing. Bernhard Löwenstein
Inzersdorferstraße 113A/ 6
A-1100 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 13.11.2009 .....
(Unterschrift)

Danksagung

"Leider lässt sich eine wahrhafte Dankbarkeit mit Worten nicht ausdrücken."
(Johann Wolfgang von Goethe, 1749-1832)

Trotzdem möchte ich dies versuchen und hiermit meiner Familie ein großes Dankeschön aussprechen! In so manch schwieriger Zeit stand sie mir bei und motivierte mich in Bezug auf mein Studium, dieses erfolgreich zu Ende zu bringen. Ausgleich fand ich dabei beim gemeinsamen, konstruktivistischen Entdecken und Erforschen der Welt mit meinem Neffen David und meiner Nichte Katharina. Es ist schön, dass es euch gibt!

"You get me job, you get me place to live in your building, you very very good man."
*(Brian George [als Babu Bhatt in Seinfeld], * 1952)*

Auch wenn dieses Zitat aus der Sitcom Seinfeld weder grammatikalisch noch sprachlich perfekt ist, so lässt sich zwecks Danksagung an meinen Ex-Chef, Mentor, Förderer und Freund Reinhard Irsigler wohl kaum ein treffenderes und besseres finden. Er begegnete mir stets mit Großzügigkeit und hatte außerdem immer großes Vertrauen in meine technischen Fähigkeiten. Dafür möchte ich auf diesem Wege sehr herzlich danken!

"Im Himmel ist ein Engel nichts Besonderes."
(George Bernard Shaw, 1856-1950)

Auf Erden aber schon, und so musste ich viele Semester studieren, bis mir ein solcher Engel in der Gestalt meiner Betreuerin an der TU Wien erschien: Monika Di Angelo. Aus didaktischer Sicht stellt sie eine der strahlenden Erscheinungen an dieser Universität dar. Nicht nur dafür, sondern vor allem auch für die ausgezeichnete Zusammenarbeit möchte ich mich hiermit herzlichst bedanken!

Abstrakt

Glaubt man dem Slogan eines der führenden Süßwarenhersteller Europas, dann bedarf es lediglich "Spiel, Spaß und was zum Naschen", um Kinder glücklich zu machen. In Anlehnung an diesen Werbespruch lässt sich ein wünschenswertes Motto für den modernen Schulunterricht ableiten. Und zwar sollte den Schülern dort "Spiel, Spaß und was zum Lernen" geboten werden.

LEGO Mindstorms folgt ganz dieser Devise – es eignet sich zum Spielen, macht den Kindern Spaß und sie lernen dabei auch etwas. Weiters unterstützt es den Unterrichtenden dabei, seine Schüler für den zu vermittelten Lernstoff zu begeistern. So ist es wenig verwunderlich, dass die unterschiedlichen Roboterbaukästen dieser Serie immer öfter in den Klassenzimmern anzutreffen sind, um während der Unterrichtseinheit als Lernhilfe zu fungieren.

Die Einführung einer neuen Technologie geht oft mit der Hoffnung einher, dass jene die Branche revolutionieren wird. Auch das Paradigma der Objektorientierung ließ eine solche Zuversicht aufkommen. Es dauerte dann allerdings Jahrzehnte, bis diese in der Industrie weite Verbreitung fand und noch länger, bis deren Vermittlung ein Thema in den Bildungsanstalten wurde. Mittlerweile wird jedoch an vielen Schulen der Versuch unternommen, den Schülern auf mehr oder weniger kreative Art und Weise die grundlegenden Prinzipien der Objektorientierung näherzubringen.

Diese Arbeit erforscht das Potential von LEGO Mindstorms im Bereich der Vermittlung der objektorientierten Konzepte. Dazu werden unterschiedliche konkrete Lehransätze betrachtet und hinsichtlich ihrer Eignung für den Schulunterricht analysiert. Untersucht wird auch, welche der Basiskonzepte sich mit Hilfe von LEGO Mindstorms besonders gut einführen lassen und welche Vorteile sich gegenüber anderen Ansätzen durch dessen Verwendung ergeben. Weiters wird nach einer passenden Sprache und Umgebung für die Roboterprogrammierung Ausschau gehalten.

Abstract

Believing the slogan of one of the leading confectionery manufacturer in Europe, it only needs "fun and games, and something to nibble" to make children happy. Following this advertising slogan a desirable motto for the modern school education can be derived. Namely, "fun and games, and something to learn" should be offered to the pupils.

LEGO Mindstorms follows entirely this device – it is well suited for playing, brings joy to kids and let them also learn something in that way. Furthermore, it supports the teacher in the process of inspiring its pupils for the mediated teaching subject. Hence, it's not surprising that the diverse robotics construction kits of this series can be met increasingly in the class rooms to serve as learning aid during the teaching unit.

The introduction of a new technology often involves the hope that it will revolutionize the branch. Also, the paradigm of object-orientation let rise such a confidence. But it took decades until it found wide propagation in industries and even longer until its teaching became an issue in the educational establishment. However, meanwhile at many schools the attempt is undertaken to familiarize the pupils in a more or less creative way with the basic principles of the object-orientation.

This work investigates the potential of LEGO Mindstorms in the field of teaching of bject-oriented concepts. For this purpose different, specific teaching approaches are considered and analyzed in terms of their suitability for use in school education. It will be also examined which of the basic concepts can be well introduced via LEGO Mindstorms and which advantages result from its use compared to other approaches. Furthermore, it will be watched out for an adequate language and environment used for programming the robots.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Die folgende Tabelle enthält alle Abkürzungen, die in der Arbeit nach erfolgter Erklärung zumindest noch ein weiteres Mal verwendet werden oder ohne eine solche zum Einsatz gelangen, wie dies bei Maßeinheiten der Fall ist:

API	Application Programming Interface
CASE	Computer-Aided Software Engineering
CD	Compact Disc
COOL	Comprehensive Object-Oriented Learning
dB(A)	Dezibel(A)
DVD	Digital Versatile Disc
FLL	FIRST LEGO League
IT	Informationstechnologie
JVM	Java Virtual Machine
KB	Kilobyte
LCD	Liquid Crystal Display
MHz	Megahertz
OOA	Objektorientierte Analyse
OOD	Objektorientiertes Design
OOM	Objektorientierte Modellierung
OOP	Objektorientierte Programmierung
UML	Unified Modeling Language
USB	Universal Serial Bus
WRO	World Robot Olympiad

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Objektorientierung hat, nachdem sie Jahrzehnte auf den großen Durchbruch warten musste, die Softwarebranche in den letzten Jahren massiv geprägt und ist heute aus dem Modellierungs- und Entwicklungsbereich nicht mehr wegzudenken. So verwundert es nicht, dass die Vermittlung dieses Paradigmas mittlerweile auch an vielen Schulen ihren Platz im Informatikunterricht erhalten hat. Auch wenn die objektorientierte Denkweise gut mit dem menschlichen Denken, Erkennen und Problemlösen harmoniert, so gilt die Vermittlung ihrer Basiskonzepte als alles andere als einfach.

LEGO Mindstorms ermöglicht Kindern, Roboter aus Legosteinen zusammenzubauen und diese dann anschließend zu programmieren. Die jeweiligen Baukästen dazu enthalten neben den LEGO Technic-Elementen einen programmierbaren Baustein sowie unterschiedliche steuerbare Aktoren und abfragbare Sensoren. Zur Programmierung stehen verschiedene Sprachen und Umgebungen zur Verfügung. LEGO Mindstorms folgt ganz dem konstruktivistischen Gedanken und motiviert die Schüler durch sein direktes Feedback zum selbstständigen Erforschen der technischen Welt. Wenig überraschend fanden diese Roboterbaukästen recht schnell den Weg in die Klassenzimmer und kommen dort vorrangig im Rahmen diverser Projekte des schulischen Programmierunterrichts zum Einsatz.

Die Idee ist nun, LEGO Mindstorms zur Vermittlung der objektorientierten Konzepte zu verwenden. Konkret soll in der Arbeit aber diesbezüglich kein neuer Ansatz entwickelt und vorgestellt werden, sondern es soll lediglich untersucht werden, ob sich die Roboterbaukästen dazu eignen, Schülern das Paradigma der Objektorientierung und deren Grundprinzipien näherzubringen. Konkret geht es hierbei um eine Auseinandersetzung mit den folgenden Fragen:

- Eignet sich LEGO Mindstorms grundsätzlich zur Vermittlung der objektorientierten Konzepte im Schulunterricht?
- Welche Vorteile ergeben sich aus der Verwendung von LEGO Mindstorms gegenüber anderen Ansätzen? Sind auch Nachteile festzustellen?
- Welche der Basiskonzepte lassen sich damit besonders gut einführen und erklären? Auf welche trifft dies nicht zu?
- Die Verwendung welcher Programmiersprache und -umgebung empfiehlt sich für den objektorientierten Anfängerunterricht?

Zur Klärung dieser Fragenstellungen erfolgt einleitend eine Einführung in die wesentlichen Gebiete der Didaktik der Informatik (Kapitel 2). Danach werden dann unterschiedliche Bereiche der Objektorientierung betrachtet (Kapitel 3). Eine große Bedeutung für diese Arbeit kommt dabei der Ausarbeitung der Basiskonzepte dieses Paradigmas zu. Im Anschluss daran wird LEGO Mindstorms vorgestellt (Kapitel 4), wobei die Programmierung der Roboter im Vordergrund steht. Anschließend folgt eine Auseinandersetzung mit allgemeinen Themen, die mit der Vermittlung der objektorientierten Konzepte in Zusammenhang stehen (Kapitel 5), ehe verschiedene bestehende Vorgehensweisen dazu vorgestellt werden, wobei nur ein Teil davon LEGO Mindstorms als Lehr- und Lernhilfe nutzt (Kapitel 6). Auf Basis dieser bisherigen Ausarbeitungen wird dann zum Schluss die Analyse vorgenommen (Kapitel 7).

2 Didaktik der Informatik

Dieses Kapitel gibt eine Einführung in die Begrifflichkeiten rund um die Didaktik der Informatik, betrachtet die historische Entwicklung des Informatikunterrichts an Schulen und beschreibt die unterschiedlichen lerntheoretischen Konzepte, die didaktischen Modelle und Prinzipien sowie Lehr- und Lernmethoden. Weiters informiert es auch über die Herausforderungen, die im Informatikunterricht an den Lehrenden gestellt werden.

2.1 Einführung

Um den Begriff der Didaktik der Informatik zu klären, sollen im ersten Schritt die beiden Hauptwörter einzeln definiert werden. Danach wird dann in den interdisziplinären Begriff der Didaktik der Informatik eingeführt.

2.1.1 Didaktik

Das Wort Didaktik kommt nach [Humb06] (S. 3 ff.) vom griechischen διδακτειν (didáskein), das so viel wie lehren, unterrichten bzw. belehrt bedeutet. Im deutschsprachigen Raum wurde der Begriff der Didaktik von Johann A. Comenius (1592-1670, siehe [@wikia]) geprägt, der diesem Wort erstmals im pädagogischen Sinne eine inhaltliche Bedeutung gab ("Lehrkunst") und den Grundstein für einen stufenweisen Aufbau des Unterrichtsprozesses legte.

Als Unterrichtswissenschaft beschäftigt sich die Didaktik damit, wie die Handlungen im Unterricht auf verschiedenen Ebenen mit unterschiedlicher Praxisnähe durch Reduktion der Komplexität planbar und kontrollierbar gemacht werden können. Das Ziel der Didaktik lässt sich verkürzt mit der Frage "Was soll gelehrt werden?" beschreiben. Davon ist die Methodik zu unterscheiden, die der Frage "Wie soll gelehrt werden?" nachgeht. [Jank02] (S. 16) sieht die Beantwortung der Frage "Wer, was, von wem, wann, mit wem, wo, wie, womit und wozu soll gelernt werden?" als Aufgabe der Didaktik an.

Die unterschiedlichen Bestimmungen von Didaktik lassen sich gemäß [Korn04] (S. 41 ff.) durch die folgenden, zum größten Teil von Wolfgang Klafki (* 1927, siehe [@wikib]) geprägten Punkte wiedergeben:

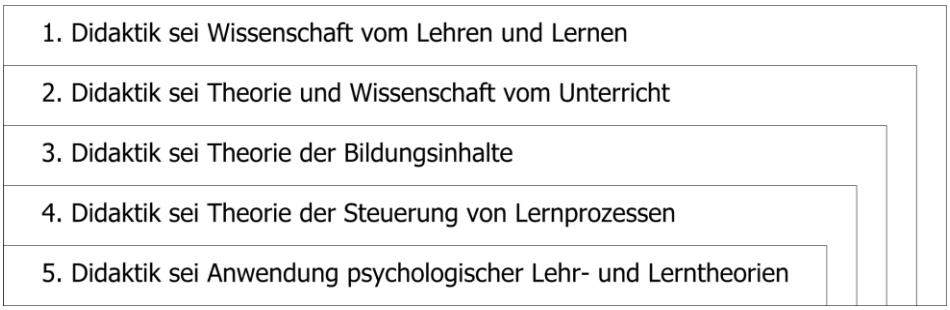


Abbildung 1: Bestimmungen von Didaktik nach Gegenstandsfeldern

2.1.2 Informatik

Der Begriff Informatik wird von [Humb06] (S. 9 f.) als Kofferwort aus Information und Automatik – andere Quellen sprechen hingegen von Information und Mathematik – erklärt. Im deutschsprachigen Raum wurde Informatik erstmals 1957 von Karl

Steinbuch (1917-2005, siehe [@wikic]) verwendet. Zu beachten ist, dass im skandinavischen Raum anstatt des Wortes Informatik eher Datalogie und im angelsächsischen Raum überhaupt computer science verwendet wird.

Aus wissenschaftlicher Sicht beschäftigt sich die Informatik mit der systematischen Verarbeitung von Informationen. Im Speziellen geht es dabei darum, wie man Information mittels Rechenanlagen automatisch verarbeiten kann.

Unterschieden werden im Wesentlichen die Teilgebiete der theoretischen, praktischen und technischen Informatik. Dazu kommen noch die interdisziplinären Disziplinen der Didaktik der Informatik, der künstlichen Intelligenz sowie der Informatik und Gesellschaft.

2.1.3 Didaktik der Informatik

Diese kurzen Beschreibungen sollten für eine Einführung reichen und erlauben die Zuwendung zum fächerübergreifenden Konzept der Didaktik der Informatik.

Nach [Baum96] (S. 10, S. 45) befasst sich diese als Fachdidaktik mit dem Unterricht in einem speziellen Fachgebiet – nämlich der Informatik – und ist zwischen der allgemeinen Didaktik einerseits und der Fachwissenschaft andererseits anzusiedeln. Ihre Aufgabe ist es, die Lehre und das Lernen auf dem Gebiet der Informatik für alle Altersstufen zu erforschen und weiterzuentwickeln.

Ihr wichtigstes Ziel ist dabei, sich mit der Gestaltung und laufenden Verbesserung des Informatikunterrichts zu beschäftigen, wobei sie sich dabei der Methoden und Konzepte der Didaktik, der Pädagogik, der Psychologie sowie der Wissenschaftstheorie bedient. [Schu04] (S. 18) nennt zusätzlich auch noch Institutionen, wie z. B. die Schule selbst als Behörde betrachtet, die auf die Fachdidaktik einwirken und stellt die Zusammenhänge wie folgt dar:

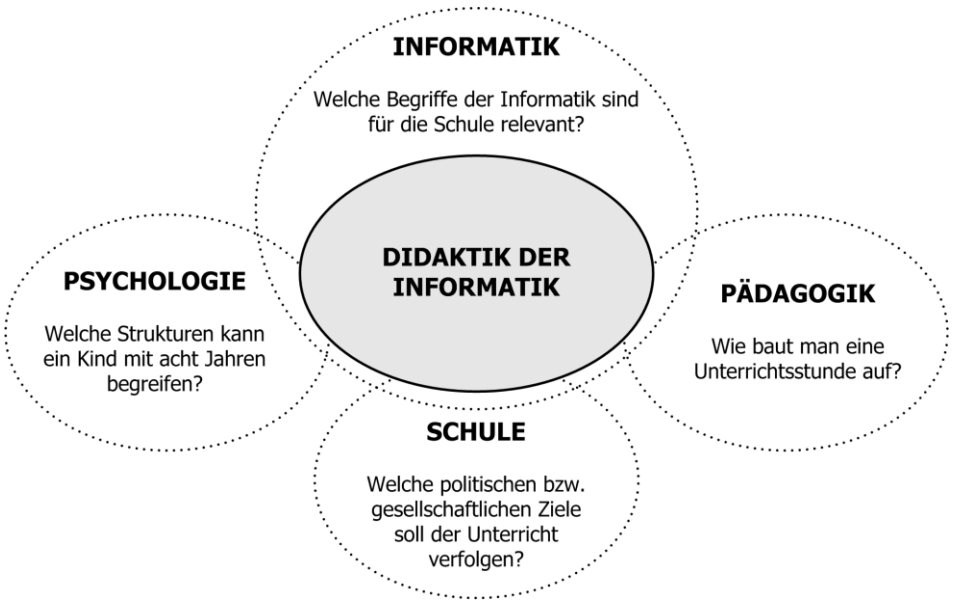


Abbildung 2: Einbettung der Didaktik der Informatik

Die Thematik lässt sich durch die Erwartungen, die an die Didaktik der Informatik gestellt werden, weiter vertiefen. [Humb06] (Seite 2 f.) beschreibt diese durch die

Formulierung einer Reihe von ausgewählten Fragen und Anforderungen, die wie folgt lauten:

- *Gehört Informatik zu den Kulturtechniken? Sollen Fragestellungen aus der Informatik in der Grundschule – oder gar im Kindergarten – thematisiert werden?*
- *Welche Inhalte und Methoden der Informatik sind allgemeinbildend?*
- *Welche Entwicklungsumgebung (für Software) soll im Unterricht in der Jahrgangsstufe 6 in der Hauptschule eingesetzt werden?*
- *Kann mit Standardanwendungen "richtiger" Informatikunterricht geplant und durchgeführt werden?*
- *Die Fachdidaktik Informatik kann nicht nur auf die Schule bezogen dargestellt werden. Es gibt auch Fragen, die sich in der Fort- und Weiterbildung und im tertiären Bildungsbereich stellen. Werden diese Fragen hier ausgespart?*
- *Wie können abstrakte Fachinhalte so aufbereitet werden, dass sie transparenter dargestellt werden?*
- *Es liegt ein konkreter fachlicher Inhalt vor. Wie kann dieser Inhalt möglichst effizient vermittelt werden? […]*
- *[…] Wie können Vorträge so gestaltet werden, dass die Zuhörerschaft besser motiviert ist, den Ausführungen zu folgen?*
- *Es gibt verschiedene Modalitäten in der Wahrnehmung von Inhalten. Wie können Rezipienten so unterstützt werden, dass eine möglichst große Bandbreite dieser Eingangskanäle "bedient" wird?*

Abschließend sei noch das von Ruth Cohn (* 1912, siehe [@wikid]) entwickelte didaktische Dreieck zu erwähnen, das die grundsätzliche Abhängigkeit zwischen dem Lehrer, seinen Schülern sowie dem zu vermittelten Stoff beschreibt und maßgeblichen Einfluss auf die Unterrichtsplanung hat. Je nach Anwendungsbereich werden in der Literatur diese drei Komponenten – im Speziellen der Stoff – unterschiedlich bezeichnet. [Jank02] (S. 55) stellt diesen Zusammenhang wie folgt dar:

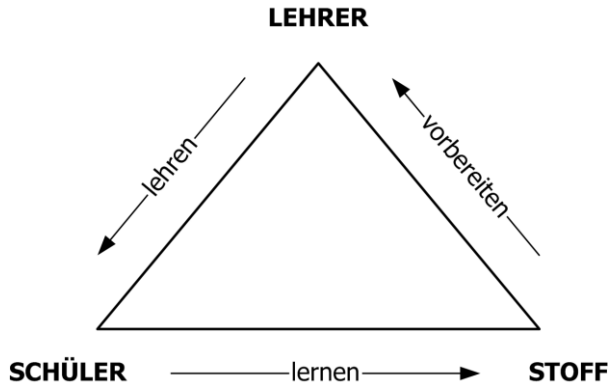


Abbildung 3: Didaktisches Dreieck zwischen Lehrer, Schüler und Stoff

Bedingt durch den zunehmenden Einsatz von neuen Medien zur Wissensvermittlung wurde das Dreieck mittlerweile zum Viereck erweitert. Dieser veränderte Sachverhalt lässt sich in Anlehnung an [Korn04] (S. 32) nun folgendermaßen wiedergeben:

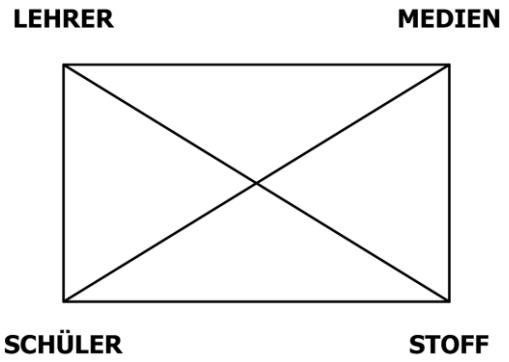


Abbildung 4: Didaktisches Viereck zwischen Lehrer, Schüler, Stoff und Medien

2.2 Geschichtliche Entwicklung des Informatikunterrichts

Obwohl die Informatik ein noch recht junges Schulfach ist, hat es bereits einige Paradigmenwechsel hinter sich gebracht. Die einzelnen Phasen nach [Baum96] (S. 112 ff.), [Hubw07] (S. 50 ff.) und [Humb06] (S. 51 ff.) werden in Folge näher beleuchtet. Zu beachten ist dabei, dass die Werke in Bezug auf die letzte Phase voneinander abweichen.

	[Baum96]	[Hubw07]	[Humb06]
1.	Hardwareorientierung		
2.	Algorithmenorientierung		
3.	Anwendungsorientierung		
4.	Benutzerorientierung		
5.	Systemorientierung		Gesellschaftsorientierung

1970 | 1975 | 1980 | 1985 | 1990 ▼

Abbildung 5: Phasen der Entwicklung des Informatikunterrichts

2.2.1 Hardwareorientierung

Die ersten Unterrichtsversuche auf dem Gebiet der Informatik, damals noch Datenverarbeitung genannt, gab es um das Jahr 1965 herum. Elektronische Rechenanlagen waren zu diesem Zeitpunkt eine noch recht neuartige Erfindung und standen am Beginn ihrer Entwicklung. Trotzdem konnten der breiten Öffentlichkeit in Sachen Rechnertechnik schon recht bald beachtliche Erfolge, wie z. B. die erste bemannte Mondlandung im Jahr 1969, präsentiert werden.

Im Unterricht standen die hardwaremäßige Vermittlung von Datenverarbeitungsanlagen ("Rechnerkunde"), der Bau digitaler Schaltnetze und -werke sowie das Programmieren an Modellrechnern im Vordergrund. Flussdiagramme wurden zur Beschreibung von Algorithmen verwendet. Die Anwendungen waren hauptsächlich im numerischen Bereich angesiedelt. Das vorrangige Lernziel war die Vermittlung der mathematisch-technischen Grundlagen der Datenverarbeitung.

2.2.2 Algorithmenorientierung

Mitte der 70er Jahre kam es dann zur Umorientierung. Im Vordergrund stand nun nicht mehr das technische Funktionieren des Rechners, sondern dessen Möglichkeiten und die Auswirkungen bzw. Grenzen des Rechnereinsatzes.

Im Unterricht rückten die Algorithmen in den Vordergrund. Den Schülern wurde nun die Formulierung und Programmierung von Algorithmen, die Analyse von Problemen mit algorithmischem Hintergrund und die Umsetzung von Algorithmen in Programme vermittelt. Des Weiteren brachte man ihnen eine systematische Vorgehensweise zur Lösung einer vorgegebenen Problemstellung bei. Die Methode umfasste dabei die folgenden Schritte: Erfassen des Problems, Entwurf eines Lösungsvorschlages, Umsetzung der Lösung, Überlegungen bzgl. Korrektheit und Diskussion möglicher Alternativen.

Man stellte allerdings bald fest, dass die vollständige Beschreibung eines Problems in Algorithmenform kaum altersgerechte Beispiele mit höherer Komplexität zuließ.

2.2.3 Anwendungsorientierung

Nachdem die Forderung aufkam, dass sich der Unterricht an konkreten Situationen des Lebens und nicht an den wissenschaftlichen Disziplinen orientieren sollte, wurde ein neuer Ansatz entwickelt, der die Lösung praktischer Probleme in den Vordergrund rückte und dabei auch die gesellschaftlichen, kulturellen und psychologischen Dimensionen der Lösungsfindung einbezog. Des Weiteren wurde die Modellbildung nun stärker hervorgehoben und die Methoden der professionellen Softwareentwicklung wurden intensiver berücksichtigt.

Neben der Überschätzung der Anwendungen blieb bei der anwendungsorientierten Methode weiterhin das bereits oben beschriebene Problem bzgl. Beispielkomplexität bestehen, da weiterhin auf die Algorithmik als Werkzeug für die Problemlösung gesetzt wurde.

2.2.4 Benutzerorientierung

Bedingt durch die zunehmende Durchdringung aller Lebensbereiche durch die Mikroelektronik und die neu aufkommenden Möglichkeiten der Vernetzung dieser Technologien wurde ein neuer Ansatz postuliert, der den Benutzer in den Mittelpunkt rückte. Dieser sah den Verzicht auf die Programmierung, die Verwendung von Anwendungsprogrammen sowie die Orientierung an der Praxis vor.

Als Ziele für den Unterricht lassen sich bei der benutzerorientierten Methode die informationstechnologische Allgemeinbildung, die Vermittlung eines sinnvollen Umgangs mit den Informationstechnologien und die Verbesserung der Fähigkeiten zur Beurteilung einzelner Anwendungen sowie zur Bewältigung von Problemen, die durch die Ausbreitung dieser Technologien entstehen, nennen.

Als problematisch stellte sich allerdings heraus, dass die Schüler nur die Programmoberfläche zu sehen bekamen, wodurch ihnen interne Details, wie die gewählten Datenstrukturen oder Problemlösungen, verborgen blieben.

2.2.5 Systemorientierung

[Baum96] (S. 114 f.) nennt als fünfte Phase den systemorientierten Ansatz, bei dem der Informatikunterricht unter dem Motto "Vom algorithmischen zum systemorientierten (bzw. synergetischen) Denken" steht. In den Mittelpunkt rückt hier die Synergetik, also die Lehre vom Zusammenwirken von Elementen.

Statt um die Implementierung einzelner, isolierter Programme geht es dabei um Systeme, deren Basis entsprechend zusammenwirkende Komponenten (Teilsysteme) sind. Weiters wirken nun bei einem Informationssystem verschiedene Wissensformen und bei der Entwicklung eines solchen Systems unterschiedliche Interessensvertreter zusammen. Der Unterricht wird beim systemorientierten Ansatz derart organisiert, dass die Schüler projektmäßig zusammenarbeiten, anstatt die Probleme einzeln zu lösen.

2.2.6 Gesellschaftsorientierung

[Humb06] (S. 52) erwähnt als fünfte Phase den gesellschaftsorientierten Ansatz, in dessen Rahmen primär die Auswirkungen der Gesellschaft auf die Informatik und umgekehrt untersucht werden. Im Mittelpunkt des Interesses stehen nun nicht mehr Einzelpersonen oder Firmen, sondern die Gesellschaft im Ganzen, für die der Nutzen von Hilfsmitteln, Methoden und Systemen untersucht werden soll.

2.3 Lernpsychologische Theorien

Die Lernpsychologie beschäftigt sich mit der Untersuchung menschlicher Lernvorgänge. Die folgende Tabelle nach [Baum94] (S. 110) gibt einen kurzen Überblick über die Charakteristika der drei wesentlichen Lernparadigmen, die dann im Anschluss ausführlicher erklärt werden:

	Behaviorismus	**Kognitivismus**	**Konstruktivismus**
Hirn ist	passiver Behälter	informationsverarbeitendes "Gerät"	informationell geschlossenes System
Wissen wird	abgelagert	verarbeitet	konstruiert
Wissen ist	eine korrekte Input-/ Output-Relation	ein adäquater interner Verarbeitungsprozess	mit einer Situation operieren zu können
Lernziele	richtige Antworten	richtige Methoden zur Antwortfindung	komplexe Situationen bewältigen
Paradigma	Stimulus-Response	Problemlösung	Konstruktion
Strategie	lehren	beobachten und helfen	kooperieren
Lehrer ist	Autorität	Tutor	Coach, (Spieler)Trainer
Feedback	extern vorgegeben	extern modelliert	intern modelliert

Tabelle 1: Charakteristika von Behaviorismus, Kognitivismus und Konstruktivismus

2.3.1 Behaviorismus

Beim Behaviorismus wird das Gehirn als eine Art Blackbox angesehen, die nach der Stimulation durch einen Reiz mit einer entsprechenden Reaktion antwortet. Wird eine solche von der Umwelt positiv verstärkt, dann erhöht dies die zukünftige Auftrittswahrscheinlichkeit dieses Verhalten.

Besondere Bekanntheit erlangte in Zusammenhang mit dem Behaviorismus der Hundeversuch von Iwan P. Pawlow (1849-1936, siehe [@wikie]). Bei diesem Experiment ließ Pawlow jedes Mal vor der Fütterung des Hundes eine Klingel ertönen. Nach einiger Zeit reagierte der Hund rein auf das Klingeln mit Speichelfluss, was davor nur bei der Fütterung der Fall gewesen war. Die daraus erlangten Erkenntnisse wurden von John B. Watson (1878-1958, siehe [@wikif]) auf die Lernpsychologie übertragen, der die Ansicht vertrat, dass sich jegliches Verhalten durch solche Reiz-Reaktions-Verknüpfungen erklären lassen würde.

Nach [Hubw07] (S. 3 f.) steht die Veränderung der Verhaltensweisen durch den Lernprozess im Zentrum des Behaviorismus, dessen Hauptziel es ist, Theorien zur Verfügung zu stellen, durch die bestimmte Reaktionen in gegebenen Situationen vorhersagbar werden. Da sämtliche psychologischen Erkenntnisse in Form von Experimenten nachweisbar sein müssen, ist die Lerntheorie allerdings auf die Erklärung beobachtbarer Phänomene beschränkt.

Geschichtlich lässt sich der Behaviorismus gemäß [@wikih] in die Phasen des klassischen, des Neo- und des radikalen Behaviorismus unterteilen, wobei letztere Strömung unweigerlich mit dem Namen Burrhus F. Skinner (1904-1990, siehe [@wikig]) in Verbindung gebracht wird.

Als konkrete Tipps für die Praxis lassen sich aus der behavioristischen Lehre ableiten: Der Lernende sollte während des Lernprozesses kontinuierlich positiv bestärkt werden und eine Bestrafung so weit wie möglich vermieden werden.

2.3.2 Kognitivismus

Als Gegenbewegung zum Behaviorismus entwickelte sich parallel dazu die kognitivistische Psychologie, die sich gemäß [@wikii] vorrangig mit den folgenden drei Fragestellungen auseinandersetzte:

- Wie strukturieren Menschen ihre Erfahrungen?
- Wie messen sie ihnen Sinn bei?
- Wie setzen sie ihre gegenwärtigen Erfahrungen zu vergangenen, im Gedächtnis gespeicherten Erfahrungen in Beziehung?

Anstatt nun nur mehr das äußere Verhalten zu untersuchen, zielte die Forschung auf die Beobachtung der strukturellen Veränderungen im Gehirn durch den Lernprozess ab. Im Mittelpunkt des Interesses standen höhere geistige Prozesse, durch die sich Vorgänge wie Auffassung, Lernen, Planung, Einsicht und Entscheidungen erklären lassen würden.

Donald O. Hebb (1904-1985, siehe [@wikij]) war einer der Pioniere auf dem Gebiet des Kognitivismus. Dieser erklärte den Lernvorgang laut [Hubw07] (S. 5 f.) mittels elektrochemischer Prozesse im Gehirn, wobei den Neuronen als Träger der elektrochemischen Impulse eine besondere Rolle zukam. Sie verbanden nämlich die Rezeptoren mit den Effektoren. Dieses Modell von Hebb stellte die Basis für die weiteren Arbeiten auf diesem Forschungsgebiet dar.

Folgende Praxistipps finden sich in der Literatur: Die Lehrinhalte sollten sinnvoll strukturiert werden und dem Lernenden auch die Zusammenhänge zwischen denselben aufgezeigt werden, wobei besonders bzgl. bereits bekanntem Wissen viele Anknüpfungspunkte aufzuzeigen sind.

2.3.3 Konstruktivismus

Gemäß [@wikik] unterliegt beim Konstruktivismus jegliches Lernen bestimmten Konstruktionsprozessen, auf die sinnesphysiologische, neuronale, kognitive und soziale Prozesse einwirken. Jeder Lernende konstruiert bzw. erschafft sich demnach durch den Lernprozess seine eigene Repräsentation der Welt. Dies hat zur Folge, dass der Schüler und seine Erfahrungen stark darauf einwirken, was er erlernt.

In den letzten Jahren haben sich unterschiedliche Strömungen dieser Lerntheorie entwickelt. [Hubw07] (S. 10 f.) nennt dazu die folgenden vier Ansätze:

- Situierte Erkenntnis (Situated Cognition): Der sozialen Umgebung und dem inhaltlichen Kontext werden bei dieser Richtung eine besondere Bedeutung zugemessen.
- Narrativer Anker (Anchored Instruction): Das Lernwissen wird hierbei mit einer anschaulichen und interessanten Geschichte, dem sogenannten narrativen Anker, verknüpft. Der Lehrstoff wird dadurch doppelt im Gedächtnis fixiert.
- Kognitive Flexibilität (Cognitive Flexibility): Komplexe Inhalte in wenig strukturierten Bereichen werden bei diesem Ansatz aus möglichst unterschiedlichen Perspektiven beleuchtet bzw. in verschiedenen Zusammenhängen wiedergegeben. Dies führt dazu, dass im Gehirn mehrere Zugänge zum Gelernten geschaffen werden.
- Kognitive Handwerkslehre (Cognitive Apprenticeship): Die traditionelle Handwerkslehre dient dabei als Vorbild. Ein Experte (Meister) demonstriert dem Lernenden (Lehrling) die korrekte Vorgehensweise und Problemlösung. Durch entsprechendes Nachmachen erprobt der Schüler im Anschluss daran das Ganze selbst.

Als praktische Tipps für den Unterricht lassen sich folgende anführen: Der Schüler sollte sich nicht nur aktiv mit dem Stoff auseinandersetzen, sondern auch die Problemlösungsverfahren möglichst selbstständig erarbeiten müssen. Des Weiteren sollte der Lehrer vorrangig in beratender Funktion auftreten und ausreichend Zeit für die Beschäftigung mit dem Stoff einplanen.

2.4 Didaktische Modelle

Im Laufe der Zeit wurden verschiedene didaktische Modelle entwickelt, die theoretische Anleitungshilfen für die Durchführung des Unterrichts darstellen und hinsichtlich Lehren und Lernen auf unterschiedliche Aspekte abzielen. Die bedeutendsten Vertreter werden nun vorgestellt.

2.4.1 Bildungstheoretische Didaktik

Nach [Hubw07] (S. 25 f.) steht beim bildungstheoretischen Ansatz – man nennt diesen auch die Göttinger Schule – die bildende Begegnung des Menschen mit der kulturellen

Realität im Vordergrund. Die Wirklichkeit erschließt sich dadurch dem Menschen und umgekehrt, was den Prozess der kategorialen Bildung bedingt.

Das Hauptziel bei diesem Modell ist die Vermittlung von Allgemeinbildung, weshalb die Auswahl der Inhalte nicht mehr von den Fachwissenschaften selbst durchgeführt werden kann. Die Methodik tritt in den Hintergrund, ins Zentrum des Interesses rücken hingegen die Lehrinhalte. Es geht also vorrangig um die Frage "Was ist zu lehren?".

Für die Festlegung der Unterrichtsfächer sind die folgenden fünf Fragen gemäß dem oben genannten Werk wesentlich:

- *Welche exemplarische Bedeutung hat der Unterrichtsgegenstand?*
- *Wie bedeutend ist er für die Gegenwart?*
- *Welche Bedeutung für die Zukunft lässt sich vermuten?*
- *Wie ist die Struktur des Inhalts?*
- *Wie steht es mit der unterrichtlichen Zugänglichkeit?*

2.4.2 Lerntheoretische Didaktik

Bei diesem didaktischen Ansatz, der auch als Berliner Didaktik in die Geschichte einging, wird gemäß [Hubw07] (S. 26 f.) die Didaktik als Theorie des Lehrens und Lernens aufgefasst. Vorrangig geht es hierbei nun um die Unterrichtsmethodik, also um die Klärung der Frage "Wie soll man den Stoff vermitteln?".

Von essentieller Bedeutung ist bei der lerntheoretischen Didaktik die Planung des Unterrichts. Eine wichtige Rolle spielen dabei die unterschiedlichen Bedingungsfelder und Entscheidungsfelder. Der Lehrende muss demnach seine Entscheidungen unter Beachtung bestimmter Voraussetzungen treffen.

Folgende Bedingungsfelder werden unterschieden:

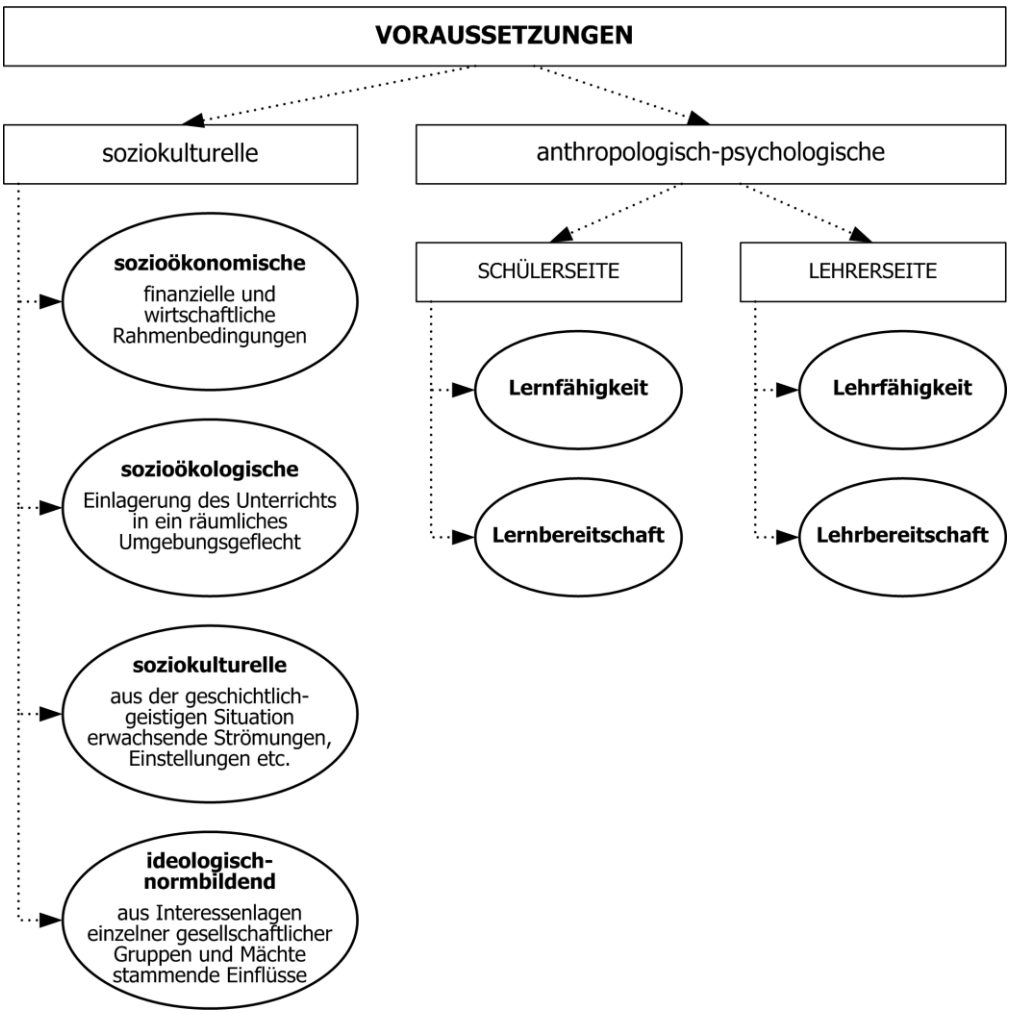


Abbildung 6: Bedingungsfelder der lerntheoretischen Didaktik

Bei den Entscheidungsfeldern wird zwischen den in Folge angeführten differenziert:

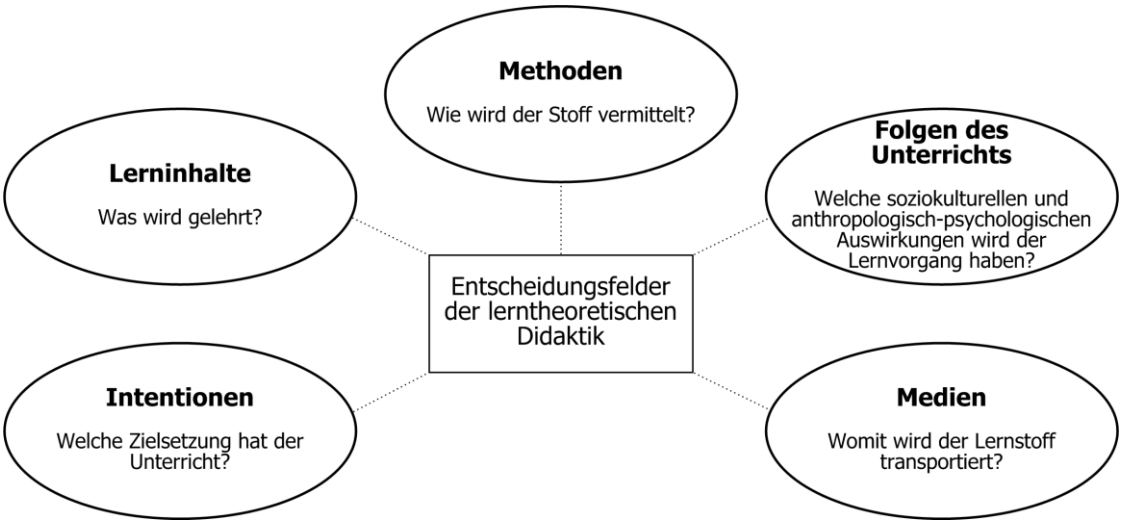


Abbildung 7: Entscheidungsfelder der lerntheoretischen Didaktik

2.4.3 Curriculare Didaktik

[@wikil] nennt als weiteres didaktisches Modell die curriculare Didaktik, die auch als lernzielorientierter Unterricht bezeichnet wird.

Wesentliche Begriffe in Zusammenhang mit diesem Ansatz sind der Lehrplan (Curriculum) und das Lernziel. Ersterer legt den Aufbau und Ablauf der Unterrichtseinheiten fest, zweiterer definiert die durch den Unterricht gewünschte, beobachtbare Verhaltensänderung des Schülers. Wichtig dabei ist, dass die Ziele präzise festgelegt werden, denn nur so kann eine effektive Methodenauswahl erfolgen und am Ende der Lernerfolg gemessen und entsprechend beurteilt werden.

Der Prozess beim curricularen Modell unterteilt sich in die drei folgenden Teilprozesse:

Abbildung 8: Teilprozesse der curricularen Didaktik

2.4.4 Informationstheoretisch-kybernetische Didaktik

Nach [Baum96] (S. 26) bedient sich dieser Ansatz bestimmter Methoden der Kybernetik und der Informationstheorie und bemüht sich um eine Automatisierung der Lehrstrategien. Die Didaktik wird dabei auf die reine Methodik reduziert.

Im Grunde werden die Schüler beim informationstheoretisch-kybernetischen Modell als informationsverarbeitende Systeme angesehen, die durch entsprechende Verfahren dahingehend beeinflusst werden sollen, dass sie das im Lehrziel definierte Verhalten annehmen.

[Hubw07] (S. 28) stellt die Zusammenhänge wie folgt dar:

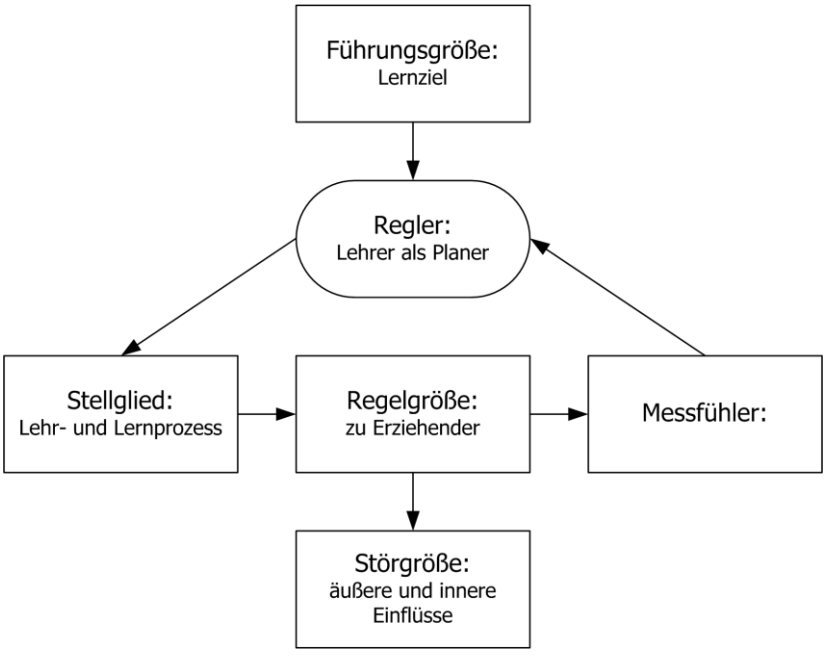


Abbildung 9: Regelkreis der informationstheoretisch-kybernetischen Didaktik

2.4.5 Konstruktivistische Didaktik

Bei der konstruktivistischen Didaktik wird laut [@wikim] das Lernen als individueller Prozess beschrieben, bei dem das Wissen auf Basis der Wirklichkeits- und Sinnkonstruktion jedes einzelnen Lernenden selbst organisiert wird.

Man geht in diesem Zusammenhang davon aus, dass Wissen als solches nie direkt von einer auf eine andere Person übertragen werden kann, sondern dass dieses nur durch selbstständigen Aufbau bzw. Konstruktion erworben werden kann. Die Aufgabe des Lehrers besteht nun darin, eine adäquate Lernumgebung zu schaffen, in der sich die Schüler auf pragmatische, interaktive und kreative Art und Weise mit dem neuen Lernstoff auseinandersetzen können.

2.4.6 Kommunikative Didaktik

Der Unterricht wird bei diesem didaktischen Modell gemäß [Hubw07] (S. 28) als kommunikativer und edukativer Prozess angesehen.

Hinsichtlich Kommunikation werden die zum größten Teil durch den Lehrplan festgelegte Inhaltsdimension und die das Verhältnis zwischen Lehrer und Schüler beschreibende Beziehungsdimension als wesentliche Faktoren genannt. Ebenfalls starke Beachtung findet die fortschreitende Emanzipation der Lernenden gegenüber ihrem Lehrer sowie ihrer Umgebung.

[@wikin] unterscheidet in Zusammenhang mit diesem Ansatz zwischen der älteren kommunikativen, der subjektiven und der neueren kommunikativen Didaktik.

2.5 Didaktische Prinzipien

Die Lehrenden haben bei der Planung und Durchführung des Unterrichts eine Reihe von didaktischen Grundregeln zu beachten, damit dieser ein Erfolg wird. Derartige Prinzipien sollen bestimmte Handlungen als erwünscht, andere als unerwünscht kennzeichnen und haben demnach normativen Charakter.

2.5.1 Allgemeine Prinzipien

[Baum96] (S. 174 ff.) stellt eine Reihe allgemeiner Prinzipien für die Unterrichtsplanung und -durchführung vor. Sie sind nicht an ein bestimmtes Fach gebunden und lauten folgendermaßen:

- Prinzip des aktiven Lernens: Die Aneignung der Lehrinhalte sollte durch aktive Konstruktion erfolgt.
- Integrationsprinzip: Es sollten Beziehungsnetze und Sinnzusammenhänge geschaffen werden.
- Prinzip der Veranschaulichung: Bei der Vermittlung neuer Inhalte sollten nicht alle, sondern nur einzelne Elemente und Aspekte neu sein.
- Prinzip der Stabilisierung: Bereits erworbenes Wissen sollte immer wieder in neuen Zusammenhängen angewendet werden.
- Operatives Prinzip: Die Schüler sollten sich gedanklich mit den Lerninhalten auseinandersetzen müssen.

- Prinzip der Stufengemäßheit: Die Lerninhalte sollten der Entwicklungsstufe der Schüler entsprechend gewählt sein.
- Spiralprinzip: Ein Thema sollte nicht erst dann durchgenommen werden, wenn es im Unterricht vollständig abgehandelt werden kann, es sollten bereits vorher erste Schritte in diese Richtung erfolgen.
- Genetisches Prinzip: Die Organisation des Unterrichts sollte gemäß der erkenntnistheoretischen Vorgänge der Erschaffung und Anwendung des Faches erfolgen.
- Prinzip der Lebensnähe und Aktualität: Der Lernstoff sollte so aufbereitet werden, dass die Schüler Bezüge zu ihrer aktuellen Umgebung feststellen können.
- Prinzip des sachstrukturellen Aufbaus: Der Unterricht ist strukturiert aufzubauen und zwischen den einzelnen Einheiten sollten Zusammenhänge hergestellt werden.
- Prinzip der Zielvorstellung: Die Schüler sollten über das Lernziel informiert werden.
- Prinzip des individuellen Lerntempos: Der Unterricht sollte die unterschiedlichen Lerntempi der einzelnen Schüler berücksichtigen.

2.5.2 Prinzipien didaktischen Handelns

Die folgenden acht Prinzipien für das didaktische Handeln werden von [Hubw07] (S. 15 ff.) genannt:

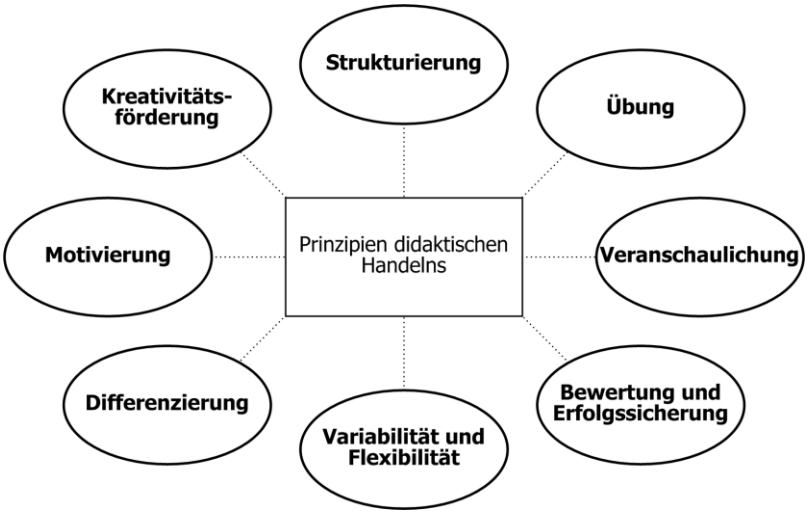


Abbildung 10: Prinzipien didaktischen Handelns

Jene werden nun nach vorhin angeführtem Werk näher charakterisiert.

2.5.2.1 Motivierung

Dass es ohne Motivation keinen Lernerfolg gibt, ist kein Geheimnis. Somit muss die Motivierung seiner Schüler das wichtigste Ziel eines jeden Lehrenden sein.

Zu unterscheiden sind in diesem Zusammenhang die Begriffe der Motivierung, der Motive sowie der Motivation, deren Zusammenwirken sich wie folgt darstellen lässt:

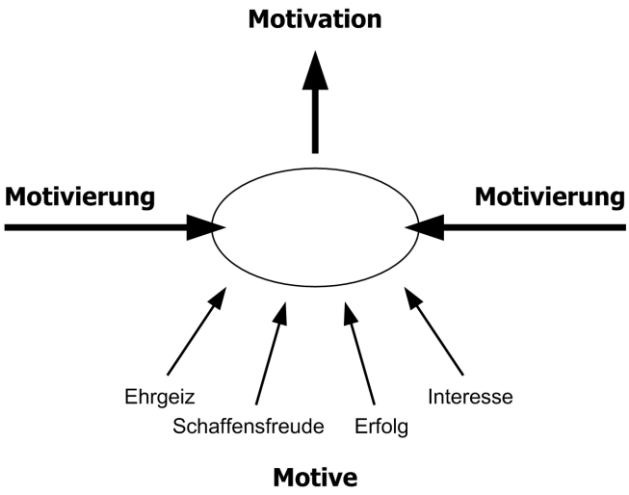


Abbildung 11: Motivierung, Motive und Motivation

Besonders zu beachten ist die Wechselwirkung zwischen der Motivation und dem Lernen. Die Motivation ist nämlich sowohl Grundlage als auch Ergebnis eines Lernprozesses. Untersuchungen dazu zeigten, dass sich die Lernleistung eines Schülers durch die lernerfolgsbedingte Motivationsverstärkung in manchen Fällen sogar verdoppelt.

Eine Einteilung der Motivation bzw. Motivierung kann nach verschiedenen Aspekten erfolgen:

- Lern- vs. Leistungsmotivation
- Intrinsische vs. extrinsische Motivation
- Eingangs- vs. Verlaufsmotivierung

2.5.2.2 Kreativitätsförderung

Kreativität ist für die Neukonstruktion von Wissen eine wesentliche Voraussetzung, weshalb deren Förderung ein wichtiges Ziel im Unterricht sein muss.

Zu beachten ist, dass Kreativität nur in einer freien und offenen Lernumgebung, die jedem Schüler auch Fehlschläge zugesteht, entstehen kann. Dem Lehrer kommt dabei die Verantwortung zu, für ein derartiges Unterrichtsklima zu sorgen.

Intensiv mit dieser Thematik setzt sich [Rome07] auseinander. Hier werden nicht nur die Kriterien, sondern auch gleich konkrete Vorschläge für einen kreativen Informatikunterricht genannt, welche wie folgt lauten:

- *Bereiten Sie die Schüler auf kreatives Tun vor […].*
- *Schaffen Sie […] Selbstvertrauen bei den Schülern.*
- *Verwenden Sie in Ihrem Unterricht Kreativitätstechniken zur Ideenfindung […].*
- *Schaffen Sie eine zündende Eingangserfahrung, um die Schüler […] zu interessieren. Knüpfen Sie an die Alltagserfahrungen der Schüler mit Informatiksystemen an […].*
- *Stellen Sie offene Aufgaben […].*

- *Helfen Sie den Schülern, sich in ihrer Aufgabe und Lösung mit eigenen Ideen selbst wiederzufinden.*
- *Schaffen Sie ein kreatives Unterrichtsklima […].*
- *Wenden Sie kreative Phasen regelmäßig und als langfristige Leitlinie im Informatikunterricht an.*

2.5.2.3 Strukturierung

In Bezug auf die Gliederung des Lernstoffes ist darauf zu achten, dass sich dem Lernenden die übergeordneten Zusammenhänge erschließen. Erreichen lässt sich dies durch inhaltliche Unterteilung, das Zerlegen in Teilschritte sowie durch das Aufzeigen von Abhängigkeiten und Abstraktionen.

2.5.2.4 Übung

Übungen verfolgen das Ziel, den gelernten Lernstoff durch wiederholte Anwendung zu festigen. Das Wissen wird auf diese Art in das Langzeitgedächtnis übertragen und dort in bereits bestehende Strukturen eingebettet.

Folgende Arten von Übungen können unterschieden werden:

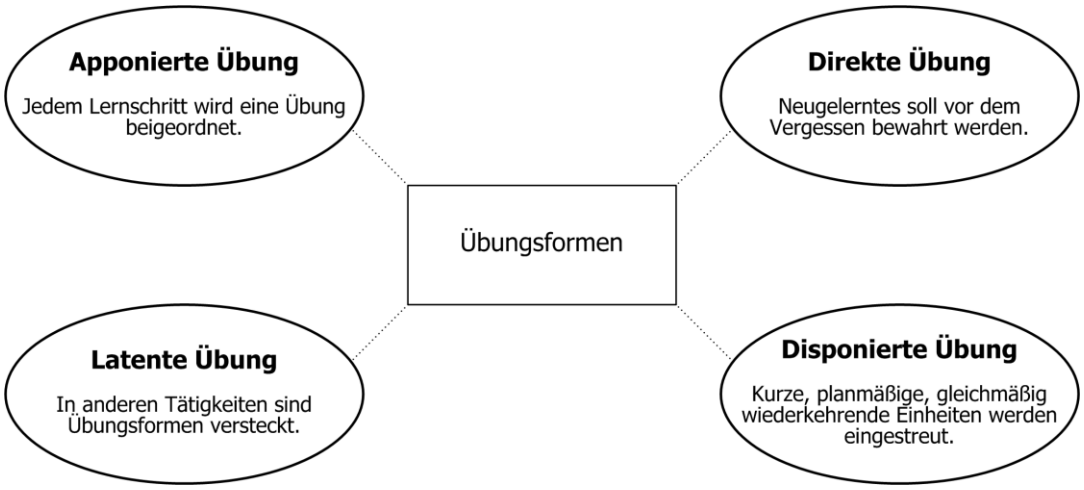


Abbildung 12: Übungsformen

2.5.2.5 Veranschaulichung

Um die Schüler bei der Verarbeitung der Unmenge von Informationen und der damit verbundenen Fülle von Reizen zu unterstützen, ist der Lernstoff anschaulich zu gestalten. Dies hilft ihnen dabei, das Wichtige vom Unwichtigen unterscheiden und Strukturen erkennen zu können.

Ganzheitliche Lernkonzepte sollten generell immer unterschiedliche Sinne ansprechen. Versuche zeigten nämlich, dass durch das Hören, Sehen und Sprechen unterschiedliche Regionen im Gehirn aktiviert werden. Auch der Einsatz verschiedener Medien empfiehlt sich, wobei eine Reizüberflutung allerdings tunlichst zu vermeiden ist.

2.5.2.6 Bewertung und Erfolgssicherung

Um den Unterrichtserfolg bewerten zu können, sind Lernzielkontrollen vonnöten. Dafür lassen sich die folgenden Punkte heranziehen:

- Fragen des Lehrers zu Aufgaben, wobei dieser auf eine Antwort bestehen sollte
- Beiträge der Schüler
- Ergebnisse von Stillarbeiten, Gruppenarbeiten oder Schülervorträgen
- Ergebnisse schriftlicher Prüfungen
- Ergebnisse von Hausaufgaben, wobei auch die nicht abgegebenen als Gradmesser dienen
- Unruhe in der Klasse, die oftmals aus Über- oder Unterforderung resultiert

2.5.2.7 Variabilität und Flexibilität

Grundsätzlich sollte der Unterricht sehr variabel und flexibel durchgeführt werden und unterschiedliche Lernmethoden und Medien zum Einsatz gelangen. Dies verhindert unter anderem, dass bei den Schülern Langeweile aufkommt oder ein bestimmter Lerntyp bevorzugt wird und unterstützt die Strukturierung.

In diesem Zusammenhang ist noch auf [Feld88] zu verweisen. Darin werden nicht nur die verschiedenen Lehr- und Lernstile vorgestellt, sondern auch passende Unterrichtstechniken für jeden Lerntypen genannt.

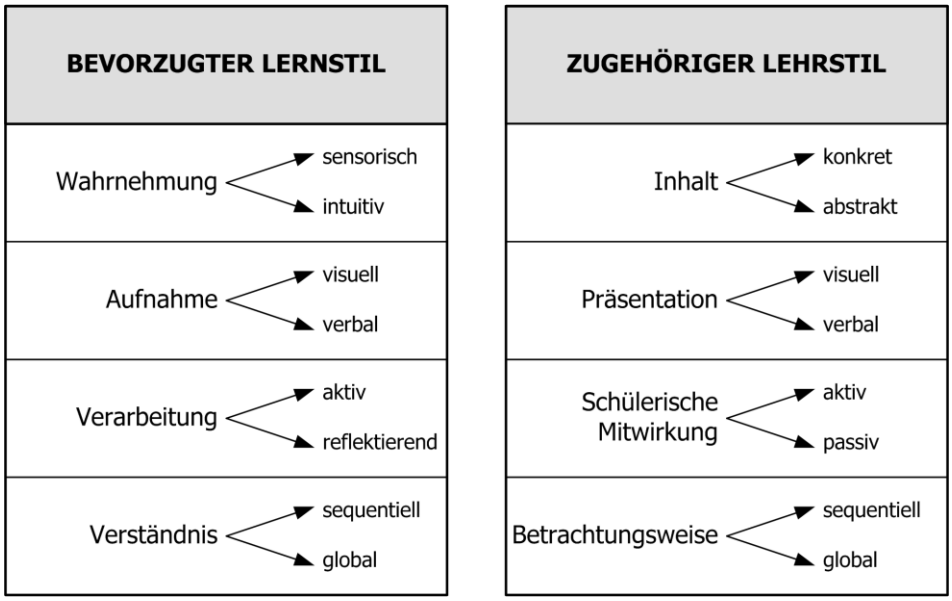


Abbildung 13: Dimensionen der Lehr- und Lernstile

2.5.2.8 Differenzierung

Hierbei wird zwischen zwei Varianten unterschieden, nämlich zwischen der äußeren und inneren Differenzierung. Bei ersterer wird der Klassenverband in Gruppen unterteilt, die getrennt voneinander unterrichtet werden. Zweitere beschäftigt sich mit der gruppeninternen Differenzierung unter einem gemeinsamen Lehrenden.

Dieses didaktische Prinzip verfolgt das Ziel, den Unterricht individueller zu gestalten, um dadurch auf die Unterschiede zwischen den Schülern einzugehen. Dieses Bemühen ist allerdings immer wieder mit organisatorischen Schwierigkeiten verbunden.

2.6 Lehr- und Lernmethoden

Bei den Lehr- und Lernmethoden geht es darum, auf welche Art und Weise der Lernstoff den Schülern nun tatsächlich nähergebracht bzw. vermittelt wird. Sie versuchen eine Antwort auf die Frage "Wie soll gelehrt werden?" zu geben.

Laut [Hart07] (S. 63 ff.) gilt es, die Unterschiede bei den Lehrenden, den Lernenden sowie den Inhalten zu beachten, weshalb eine Methodenvielfalt erforderlich ist. Bei der Methodenwahl muss außerdem berücksichtigt werden, dass die Schüler oft über unterschiedliche Vorkenntnisse verfügen.

"Informatikunterricht ist anders" – so beginnt [Humb06] (S. 75 ff.) das zugehörige Kapitel über die Methoden im Informatikunterricht und erhebt darin die Problemorientierung und -lösung zum zentralen Prinzip. Aus methodischer Sicht wird außerdem besonders das projektorientierte Lernen hervorgehoben.

2.6.1 Allgemeine Methoden

[@meth] stellt eine Vielzahl von allgemeinen Lehr- und Lernmethoden vor, von denen die meisten auch im Informatikunterricht zum Einsatz gelangen können. Zu beachten ist aber, dass während des Unterrichts oftmals die Kombination von zwei oder mehreren dieser Methoden sinnvoll bzw. ratsam ist.

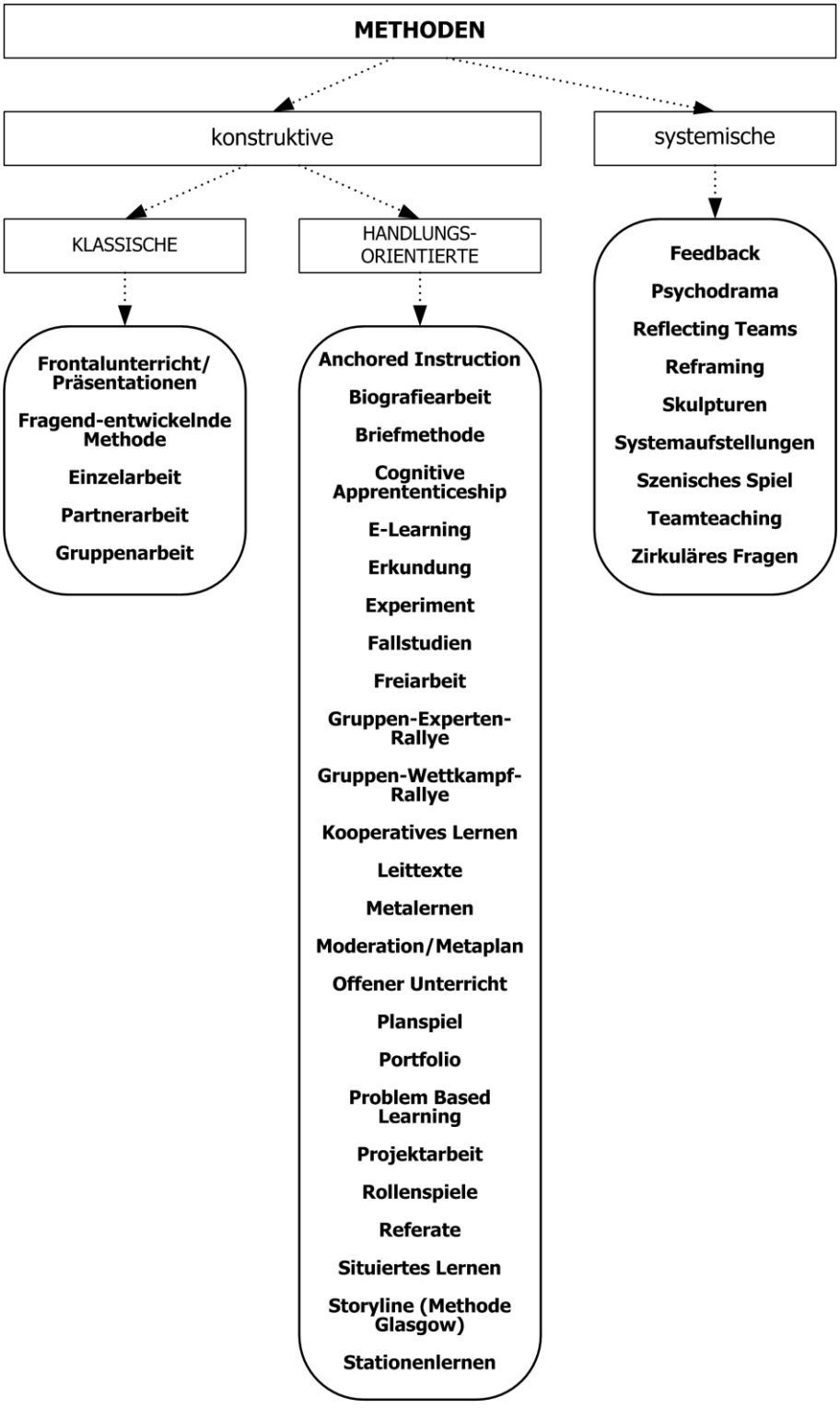


Abbildung 14: Konstruktive und systemische Methoden

Von den aufgelisteten Lehr- und Lernmethoden scheinen die folgenden für den Informatikunterricht gut geeignet zu sein, wobei diese aufgrund ihrer übersichtlichen und verständlichen Beschreibung auf der zugehörigen Webseite hier nicht näher erläutert werden:

- Frontalunterricht/Präsentationen
- Fragend-entwickelnde Methode
- Einzelarbeit

- Partnerarbeit
- Gruppenarbeit
- Cognitive Apprenticeship
- E-Learning
- Experiment
- Freiarbeit
- Gruppen-Experten-Rallye
- Gruppen-Wettkampf-Rallye
- Offener Unterricht
- Projektarbeit
- Stationenlernen
- Feedback
- Team Teaching

2.6.2 Methoden für Informatikunterricht

[Hart07] (S. 64 ff.) nennt als wesentliche Lehr- und Lernmethoden für den Informatikunterricht die folgenden, die dann anschließend in Anlehnung an das referenzierte Werk beschrieben werden sollen:

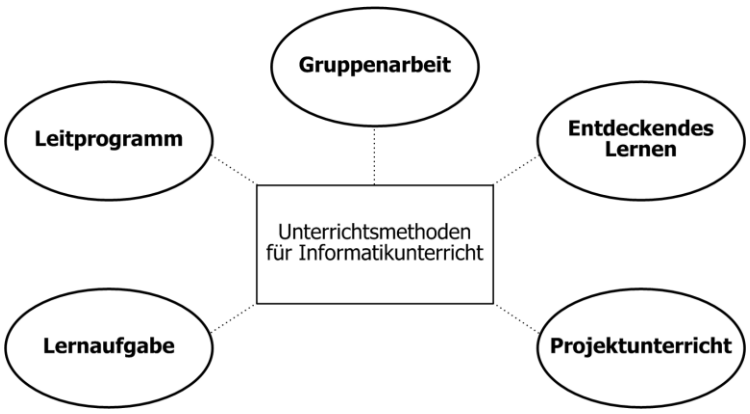


Abbildung 15: Unterrichtsmethoden für Informatikunterricht

2.6.2.1 Lernaufgabe

Bei dieser Methode, die sich für kurze Unterrichtssequenzen ausgezeichnet eignet, erklärt der Lehrer nur die wesentlichen theoretischen Aspekte des Lernstoffes und die Schüler vertiefen danach durch die selbstständige Ausarbeitung praktischer Übungen ihr Wissen.

Bei den Aufgabenstellungen gilt es, nachfolgende sechs Punkte zu beachten:

- Die Schüler müssen etwas Neues lernen.
- Die Aufgabenstellungen müssen schriftlich vorliegen.
- Es sind Hinweise zum Vorgehen anzuführen.
- Die Zeitdauer ist zu beachten.
- Die formale Antwortstruktur muss klar vorgegeben sein.

- Der Schwierigkeitsgrad der Aufgabenstellung ist zu beachten.

Da in den Lernaufgaben auf die unterschiedlichen Arbeitstempi der Schüler Rücksicht genommen wird, bedingen sie einen individualisierten Übungsablauf. Dem Lehrer kommt bei dieser Methode die Rolle eines Coachs zu, der auf die einzelnen Fragen der Schüler eingeht. Positiv lässt sich außerdem festhalten, dass durch das selbstständige Ausarbeiten der Lösungen das Selbstvertrauen der Schüler gestärkt wird.

2.6.2.2 Leitprogramm

Leitprogramme können als großer Bruder der Lernaufgabe ansehen werden. Im Mittelpunkt steht dabei das Prinzip des Mastery Learning, das sich wie folgt beschreiben lässt:

Der Lernstoff wird in einzelne, für das Selbststudium geeignete Pakete aufgeteilt. Diese werden nun vom Lernenden sequentiell durchgearbeitet, wobei er am Ende jedes Lernpaketes einen Test beim Lehrer zu absolvieren hat. Nur wenn er diesen besteht, darf er das nächste Lernpaket in Angriff nehmen. Da das Verständnis von vorangegangenem Stoff meistens die Voraussetzung dafür ist, dass der neue Stoff verstanden wird, ist diese rigide Vorgehensweise durchaus sinnvoll.

Der Aufbau eines jeden Lernpaktes folgt dabei dem folgenden Schema:

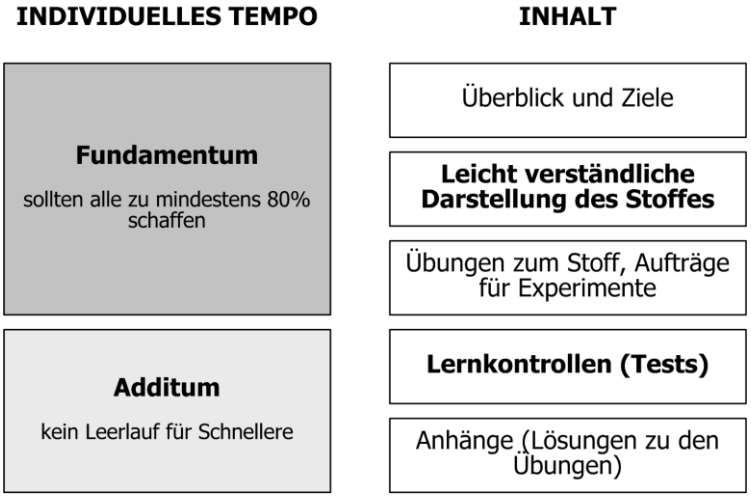


Abbildung 16: Aufbau eines Lernpaketes beim Leitprogramm

Neben der Pflicht, dem Fundamentum, gibt es also bei jedem Lernpaket auch noch die Kür, das Additum, für besonders schnelle Schüler. Es kommt somit auch bei dieser Lernmethode zu einer Individualisierung des Unterrichts und dadurch wird der Heterogenität der Lerngruppe Rechnung getragen. Für die Vermittlung schwierigerer Themen ist das Leitprogramm ebenfalls gut geeignet.

Als Nachteil lässt sich der große Aufwand für die Erstellung der Unterrichtsunterlagen nennen, der vor allem dadurch zustande kommt, dass jedes Detail vorauszuplanen ist.

2.6.2.3 Gruppenarbeit

Das vorrangige Ziel der Gruppenarbeit ist die Verbesserung der sozialen Kompetenzen der Schüler. Diesem Konzept kommt deshalb besondere Bedeutung zu, da speziell im Bereich der Informationstechnologie (IT) die Fähigkeit des Teamworks eine wichtige Kompetenz darstellt.

Als wesentliche fünf Merkmale, die auch für den Informatikunterricht interessant sind, werden für die Gruppenarbeit genannt:

- *Gruppenarbeiten fördern den Zusammenhalt unter den Lernenden. […]*
- *Die Zusammenarbeit in einer Gruppe wirkt motivierend und beschleunigt den Lernprozess.*
- *In einer Gruppe muss jedes Mitglied Verantwortung übernehmen und einen Beitrag zur Zielerreichung leisten.*
- *Gruppenarbeiten fördern Sozialkompetenzen wie Kommunikation in einem Team, Vertrauen, Entscheidungsfindung in einer Projektgruppe und Umgang mit Konflikten.*
- *Gruppenarbeiten fördern das Nachdenken über gruppendynamische Prozesse und tragen damit zur Effizienzsteigerung bei der Arbeit in Informatikprojekten bei.*

Zu beachten ist, dass der Wille zur gemeinsamen Bearbeitung eines Problems in einer Gruppe noch lange nicht den Erfolg garantiert. Besonders zu dominante Gruppenmitglieder gefährden jenen massiv.

Als besondere Ausprägungen der Gruppenarbeit lassen sich die sogenannte Puzzlemethode sowie die Partnerarbeit nennen. Beide werden im vorhin angeführten Werk ausführlich behandelt.

2.6.2.4 Entdeckendes Lernen

Im Informatikunterricht wird häufig das Paradigma der Theorievermittlung mit anschließender, praktischer Übungsausarbeitung verfolgt. Vernachlässigt werden dabei allerdings das selbstständige und kreative Arbeiten, die kritische Auseinandersetzung mit dem Stoff sowie der Austausch zwischen den Schülern.

Die Grundidee des entdeckenden Lernens – eine Form des offenen Unterrichts – ist, die im Berufsalltag geforderten Fähigkeiten zu schulen. Der Unterricht wird hierbei so gestaltet, dass jeder Schüler durch das selbstständige Auseinandersetzen mit dem Lernstoff zu neuem Wissen gelangt. Jeder generiert sich also aktiv seine Erklärungen, was wirksamer als das sture Auswendiglernen vorgetragener Theorien ist.

Für die Gestaltung des Unterrichts werden die folgenden drei Anforderungen genannt:

- Es ist auf die Offenheit des Themas zu achten.
- Das vom Lehrer zur Verfügung gestellte Material sollte vollständig sein.
- Die Aufgabenstellungen sollten mehrere Lösungen zulassen und bei der Bewertung sind alle Vorschläge und Ideen ernst zu nehmen.

2.6.2.5 Projektunterricht

Da viele Unternehmungen in der IT-Branche in Form von Projekten durchgeführt werden, ist es naheliegend, diese Methodik auch im Informatikunterricht zu üben.

Beim Projektunterricht wird ein Vorhaben gemeinsam in der Gruppe bearbeitet. Nach der anfänglichen Planungsphase machen sich die Gruppenmitglieder mit vereinten Kräften an die Umsetzung des Plans in die Realität.

Die Methode ist gut geeignet, um den Schülern einen Einblick in die Komplexität großer Informationssysteme zu gewähren. Es ist dabei allerdings zu beachten, dass aufgrund der zeitlichen Einschränkungen kaum größere und komplexere Projekte im Schulunterricht durchgeführt werden können, wodurch teilweise wichtige Aspekte bei der Planung und Umsetzung solcher Vorhaben ausgeblendet werden müssen.

2.6.3 Methoden der Programmierlehre

Die folgenden didaktischen Methoden zur Verwendung im Programmierunterricht finden sich in [Weic05]:

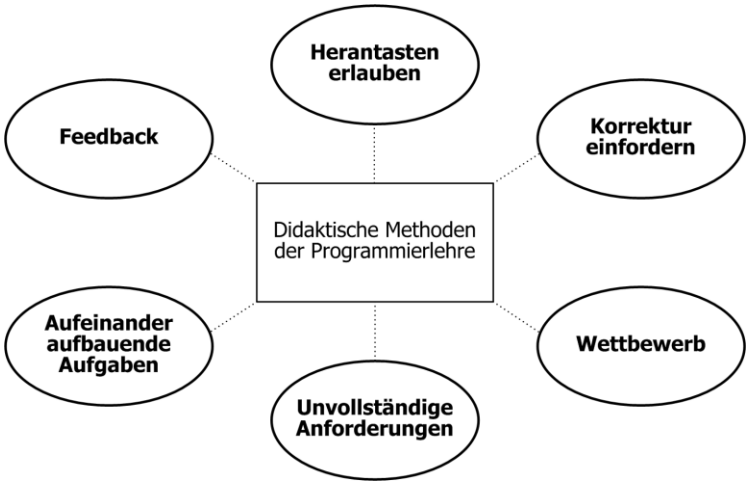


Abbildung 17: Didaktische Methoden der Programmierlehre

Es folgt nun eine detaillierte Beschreibung dieser Methoden gemäß vorhin genanntem Werk.

2.6.3.1 Feedback

Jeder Lernende erhält hierbei zu seinen ausgearbeiteten, abgegebenen Programmierlösungen ein individuelles Feedback. Damit dieser einen Zusammenhang zwischen seiner Abgabe und dem Feedback herstellen kann, muss die Rückmeldung stets sehr zeitnah erfolgen.

Um den Schüler zu motivieren, hat dabei das Feedback nicht nur Kritik, sondern auch Lob für gelungene Punkte zu enthalten. Des Weiteren sollte aufgelistet werden, welche funktionalen Tests bestanden wurden und welche nicht. Auch die verwendeten Strukturen und der Programmierstil sind vom Lehrenden entsprechend zu kommentieren.

2.6.3.2 Herantasten erlauben

Bei dieser Methode wird dem Lernenden laufend Feedback zu seinen Zwischenlösungen gegeben. Er bekommt dadurch die Möglichkeit zur schrittweisen Verbesserung bis zum Endabgabetermin.

Durch das Herantasten wird von den Schülern ein konstruktiver Umgang mit Fehlern sowie ein guter Programmierstil erlernt. Problematisch an diesem Ansatz kann sein, dass die Lösungen nicht mehr komplett durchdacht werden und bei den Schülern ein Minimaldenken einsetzt.

2.6.3.3 Korrektur einfordern

Als Alternative zum vorherigen Punkt lässt sich die folgende Variante nennen: Anstatt des kontinuierlichen Feedbacks wird jedem Schüler die Rückmeldung erst mit dem Abgabetermin gegeben. Es hat nun jeder die Möglichkeit, bis zur nächsten Deadline entsprechende Verbesserungen an seiner Lösung vorzunehmen.

Zu beachten ist hier von Lehrerseite, dass der Lernende die Korrekturen tatsächlich selbst vornimmt und diese nicht von dritter Seite durchgeführt bzw. übernommen werden. Gefördert werden durch diesen Ansatz die exakte Arbeitsweise und die Entwicklung einer eigenen Testkultur.

2.6.3.4 Wettbewerb

Hierbei veranstaltet der Lehrer einen Wettbewerb zwischen den Schülern, was meist eine zusätzliche Motivation zur Folge hat. Voraussetzung dafür ist allerdings, dass die Abgaben messbarer Natur sind, da nur so ein Vergleich und somit Wettbewerb zwischen diesen möglich ist. Konkret wird dazu z. B. die Optimierung von Algorithmen genannt.

Mit diesem Ansatz wird nicht nur die Kreativität der Lernenden gefördert, sondern auch die Teamarbeit positiv beeinflusst und die Entwicklung einer eigenen Testkultur unterstützt.

2.6.3.5 Unvollständige Anforderungen

Es ist eine wichtige Kompetenz, mit unvollständigen Anforderungen umgehen zu lernen. Die Schüler sollten deshalb bei den Programmieraufgaben auch mit solchen konfrontiert werden.

Durch die Unvollständigkeit der Anforderungen werden die Schüler nun dazu angehalten, sich kritisch mit diesen Freiräumen auseinanderzusetzen. Weiters fördert dieses Vorgehen im Falle einer Gruppenarbeit den Teamgeist, da gemeinsame Entscheidungen erforderlich sind.

2.6.3.6 Aufeinander aufbauende Aufgaben

Bei Fortgeschritteneren besteht die Möglichkeit, die Aufgaben aufeinander aufbauend zu gestalten. Die Lösungen werden dann nach jeder Zwischenabgabe unter den Schülern getauscht, wodurch nun jeder die Lösung eines anderen weiterentwickeln muss.

Dies hilft nicht nur dabei, einen sauberen Programmierstil zu entwickeln, sondern hat auch exaktere Lösungen zur Folge. Ebenfalls werden hierbei ein konstruktiver Umgang mit Fehlern erlernt und der wichtige Umgang mit fremdem Quellcode geübt.

2.7 Herausforderungen im Informatikunterricht

Beim Unterrichten des Gegenstandes Informatik kommen jede Menge Herausforderungen auf den Lehrenden zu. Einige davon sollen nun gemäß deren Darstellung in [Hart07] abgehandelt werden. Natürlich werden dazu auch immer mögliche Lösungen genannt.

2.7.1 Abstraktion und Fachbegriffe

Wesentlich für die Informatik ist der Begriff der Abstraktion. Viele Dinge lassen sich nicht anfassen – sie sind unsichtbar – und sind dadurch für die Schüler nur schwierig zu verstehen. Dazu kommen dann noch jede Menge neuer Fachbegriffe, die von so manchem Lehrenden für den Einstieg in ein neues Fachgebiet verwendet werden.

Der Abstraktion lässt sich durch entsprechende Visualisierung des Stoffes begegnen, den Schülern sollten demnach immer Bilder als Denkhilfen vermittelt werden. Auch lässt sich so manches Unsichtbare durch Hilfsmittel und -werkzeuge im Unterricht sichtbar machen und das fördert das Verständnis der Schüler für den Stoff. Bei neuen Inhalten sind, sofern möglich, immer Verknüpfungen zu bereits vorhandenem Wissen herzustellen.

2.7.2 Konzept- vs. Produktwissen

In der Informatik geht es einerseits um Konzepte, andererseits um konkrete Produkte. Dem Lehrer stellt sich nun die Frage, ob er seinen Schülern im Unterricht Konzept- oder Produktwissen vermitteln soll.

Im Grunde sollte ein passender Mittelweg gewählt werden, denn eine Fokussierung in eine der beiden Richtungen ist wenig sinnvoll. Zu beachten ist aber auf alle Fälle, dass ausreichend Bezüge zwischen den Konzepten und ihrer zugehörigen Umsetzung in den Produkten hergestellt werden.

2.7.3 Theorie vs. Praxis

Ein weiteres Dilemma ist der Spagat zwischen der Vermittlung der theoretischen Grundlagen und deren praktischer Anwendung. Die oft gewählte Vorgehensweise, dass zuerst die Theorie vorgetragen und im Anschluss dann das Ganze praktisch angewendet wird, kann aus der Sicht der Schüler wenig motivierend sein. Andererseits macht es auch oft keinen Sinn, gleich mit der Praxis zu beginnen, da einfach notwendige Grundlagen fehlen.

Der einzige Ausweg aus diesem Problem ist der Einsatz unterschiedlicher Lehr- und Lernmethoden, denn diese sorgen für Abwechslung und fordern die Schüler immer wieder neu heraus.

2.7.4 Heterogenität der Lerngruppe

Die Heterogenität innerhalb der Lerngruppe ist ebenfalls problematisch, denn die einzelnen Schüler bringen unterschiedliche Vorkenntnisse mit und auch ihre Arbeitstempi unterscheiden sich teilweise massiv.

Deshalb ist auf eine sinnvolle Methodenwahl zu achten. Konkret sind jene Methoden im Unterricht einzusetzen, welche eine Individualisierung des Unterrichts ermöglichen. Als Beispiele lassen sich hierzu die Lernaufgabe, das Leitprogramm oder das entdeckende Lernen nennen. Wesentlich ist in diesem Zusammenhang auch die Trennung zwischen Theorie und Praxis, die hinsichtlich der Dimensionen Raum, Zeit und Inhalt durchgeführt werden sollte.

2.7.5 Neue Technologien und Trends

Die Informatik ist eine Disziplin, die einem rasanten Wandel unterworfen ist. Dies hat laufend neue Versionen, Produkte und Technologien zur Folge. Dadurch kann es für die Unterrichtenden schwierig sein, die wesentlichen Inhalte im Auge zu behalten.

Trends verschwinden oft genauso schnell, wie sie aufgekommen sind. Deshalb sollten sich die Informatiklehrer eher auf die Vermittlung langlebiger Konzepte und Methoden konzentrieren. Es ist in diesem Zusammenhang jedoch zu bedenken, dass die Inhalte im Wesentlichen sowieso durch die vorgegebenen Lernziele – diese sind auch den Schülern zu kommunizieren – vorgegeben sind.

2.7.6 Probleme und Fehler

Bedingt durch Fehler in der eingesetzten Software, aber auch durch sonstige Probleme, die während des Unterrichts auftreten können, wird dessen Durchführung erschwert.

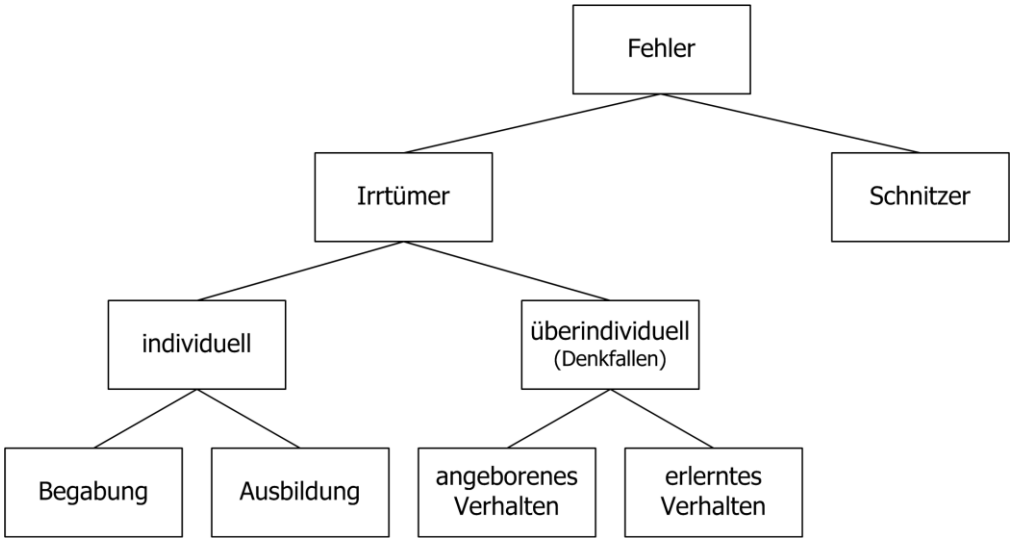


Abbildung 18: Klassifikation von Fehlerarten

Zu beachten gilt es hierbei, dass der Lehrende die Probleme nicht selbst lösen sollte, sondern seine Schüler in der Rolle eines Mentors vielmehr bei deren Lösung unterstützen sollte. Dies fördert nicht nur die Selbstständigkeit, sondern lässt auch eine Lernkultur entstehen, in der Fehler als etwas Alltägliches aufgefasst werden.

Konkret werden hierzu die folgenden Punkte genannt, die der Lehrer umsetzen kann bzw. sollte:

- Fehlerakzeptanz schaffen
- Fehlerursache finden lassen
- Problemlösungsstrategien behandeln
- Hilfe zur Selbsthilfe geben
- FAQ für häufige Probleme und Fehler verfassen

2.7.7 Komplexität von Systemen

Informationssysteme haben die Angewohnheit groß und komplex zu sein. Die Planung und Entwicklung solcher Systeme zum Zwecke der Übung im Unterricht würde allerdings dessen zeitlichen Rahmen sprengen. Jedoch gehen im Gegensatz dazu bei kleineren Systemen oftmals wichtige Aspekte verloren.

Zur Lösung dieses Problems wird der Projektunterricht vorgeschlagen, durch den sich wichtige Phasen und Teilaspekte demonstrieren lassen.

2.7.8 Weiterbildung der Lehrenden

Den bereits vorhin genannten raschen Veränderungen in Bezug auf Technologien und Produkten stehen, sofern überhaupt didaktische Konzepte zur Umsetzung der Neuerungen verfügbar sind, meist rare und oftmals teure Fortbildungskurse für Lehrende gegenüber.

Zur Lösung dieses Problems wird vorgeschlagen, dass sich Informatiklehrer gezielt über die aufkommenden Trends informieren sollten, damit sie am Ball bleiben.

3 Objektorientierung

Dieses Kapitel gibt eine kurze Einführung in die Objektorientierung und beschreibt deren historische Entwicklung. Weiters werden die objektorientierten Konzepte und die Phasen des objektorientierten Entwicklungsprozesses vorgestellt. Abschließend wird auf die konkrete Umsetzung der Konzepte in der Programmiersprache Java eingegangen.

3.1 Einführung

Zum Einstieg sollen der Begriff der Objektorientierung charakterisiert und die in der Literatur genannten Pro und Kontras in Zusammenhang mit diesem Paradigma beleuchtet werden.

3.1.1 Objektorientierung

Bei der Objektorientierung steht nach [Punt08] (S. 37) wenig überraschend der Begriff des Objektes im Mittelpunkt. Ein solches Objekt fasst zusammengehörige Daten und Methoden zu einer Einheit zusammen, wodurch ein Algorithmus nun meist auf unterschiedliche Objekte bzw. Klassen verteilt wird. Im Unterschied zur prozeduralen Programmierung stehen die verarbeitenden Informationen deshalb nicht mehr nur an einer Stelle im Programm.

Die Vorgänge in der realen Welt stellen gemäß [Poet09] (S. 2 ff., S. 15 ff.) die Grundlage für die Objektorientierung dar. Sie sollen mittels dieses Paradigmas entsprechend softwaretechnisch modelliert werden. Genauso wie es im richtigen Leben unterschiedliche Akteure gibt, die entweder Aufträge erteilen oder durchführen, gibt es in der Objektorientierung die Objekte, denen diese Aufgaben in der virtuellen Welt zukommen. Solche Objekte sind selbstständige Ausführungseinheiten, haben eine Identität und können während der Laufzeit des Programms dynamisch erzeugt bzw. gelöscht werden. In ihrer Gesamtheit bilden sie das gewünschte Programm.

Wesentlich bei der Objektorientierung ist die Trennung zwischen Auftragserteilung und -durchführung. Will jemand in der Realwelt eine Aufgabe erledigen lassen, dann muss er sich jemanden suchen, der einen solchen Job ausführen kann. In der Objektwelt läuft dies analog ab, wobei jedes Objekt Nachrichten versenden bzw. empfangen und verarbeiten kann. Durch den Empfang einer Nachricht kann es zu einer Zustandsänderung im Objekt kommen. Genauso gut können aber auch andere Objekte erzeugt, gelöscht oder benachrichtigt werden.

Zu erwähnen ist außerdem noch der Begriff der Klassifikation, der in der modellierten Welt eine wichtige Rolle spielt. Was im Geschäftsleben Branchen sind, sind bei der Objektorientierung die Klassen: Sie legen die Gemeinsamkeiten der ihr zugehörigen Objekte fest und lassen sich hierarchisch organisieren, was zu Klassenhierarchien, wie im folgenden Beispiel, führt:

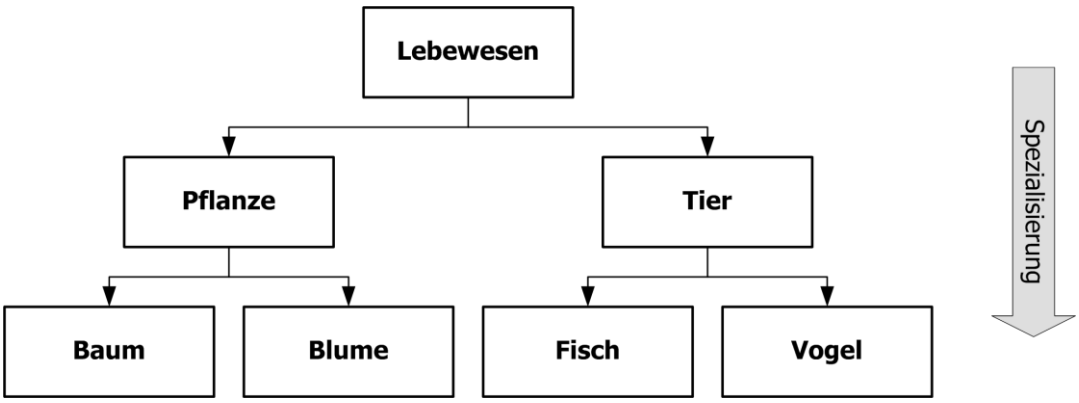


Abbildung 19: Klassenhierarchie für Lebewesen

Unweigerlich gelangt man dadurch zum Begriff der Vererbung, die ein wesentliches Konzept der Objektorientierung darstellt. Wird eine Klasse von einer anderen abgeleitet, dann erbt sie automatisch deren Eigenschaften und Methoden.

Nach [Budd91] (S. 15) sind die folgenden Punkte für die Objektorientierung charakteristisch:

- Objektorientierung bedingt eine neue Art zu denken und ist nicht nur das Hinzufügen ein paar neuer Features zu einer Programmiersprache.
- Ein Programm besteht dabei aus einer Menge autonomer Agenten, den sogenannten Objekten.
- Jedes dieser Objekte kapselt einen Zustand (Daten) und ein Verhalten (Methoden).
- Das Verhalten eines Objektes wird durch die Klasse bestimmt, der es angehört.
- Beim Empfang einer Nachricht führt das Objekt eine bestimmte Methode aus und zeigt dadurch der Außenwelt ihr Verhalten.
- Klassen unterstützen das Prinzip der Vererbung, wodurch sie sich in Form eines hierarchischen Vererbungsbaumes darstellen lassen.
- Objektorientierung ermöglicht durch die Reduzierung der Abhängigkeit zwischen den Komponenten den Bau von wieder verwendbaren Systemen.
- Durch Objektorientierung wird ein höherer Level an Abstraktion erreicht.

Zum Abschluss sei noch gemäß [Witt92] (S. 200) auf den Unterschied zwischen den beiden Begriffen objektbasiert und objektorientiert hingewiesen. Unterstützt eine Programmiersprache lediglich das Konzept des Objektes, dann gilt sie als objektbasiert. Um tatsächlich als objektorientiert klassifiziert werden zu können, muss sie alle wesentlichen objektorientierten Konzepte umsetzen.

3.1.2 Pro und Kontras

Wie sehr viele neue Technologien und Paradigmen hat auch die Objektorientierung anfangs sehr viele Erwartungen geweckt, die sich dann nur teilweise erfüllen konnte. Deshalb folgt an dieser Stelle eine Auseinandersetzung mit den Pro und Kontras.

Laut [@leara] sprechen viele Gründe für den objektorientierten Ansatz:

- Objektorientierung ist natürlicher: Der menschlichen Art und Weise des Denkens wird bei der Entwicklung von Programmen näher entsprochen.
- Objektorientierung ist ganzheitlich: Die Trennung zwischen Datentypen und Algorithmen wird großteils aufgehoben. Von Beginn an kommen entsprechende Entwurfs- und Dokumentationstechniken zum Einsatz.
- Objektorientierte Softwareentwicklung ist konsequent: Der Prozess umfasst die objektorientierte Analyse, das objektorientierte Design und die objektorientierte Programmierung, wobei in allen Phasen die gleichen Techniken verwendet werden.
- Objektorientierung ist zeitsparend: Die Wiederverwendung bestehender Klassen reduziert die Entwicklungszeit.
- Objektorientierung erhöht die Wiederverwendbarkeit und macht flexibel: Die Vererbung ermöglicht die Anpassung bereits entwickelter Klassen an neue Situationen.
- Objektorientierte Systeme sind leichter wartbar und erweiterbar: Die Aufteilung des Quellcodes auf Objekte bzw. Klassen bedingt eine bessere Wartbarkeit.
- Objektorientierung ist lebensnah: In der Wirtschaft wird größtenteils objektorientiert entwickelt.

Auch [Smit93] (S. 199 ff.) erkennt in der Anwendung der objektorientierten Programmierung klare Vorteile gegenüber der herkömmlichen prozeduralen Vorgehensweise. Nicht nur die Wiederverwendbarkeit des Codes wird deutlich verbessert, sondern auch das Lokalisieren von Änderungen vereinfacht, da diese meist nur lokal bzgl. einer Klasse durchzuführen sind. Des Weiteren erzwingt der objektorientierte Ansatz zwangsläufig einen besseren Entwurf, da bereits im Vorfeld über die Klassenhierarchie nachgedacht werden muss. Die Erweiterbarkeit der Sprache stellt einen weiteren Vorteil dar, wodurch die unerwünschten Grenzen zwischen den vorgegebenen und den eigens hinzugefügten Sprachelementen verschwinden. Dies alles bedingt einen deutlich schnelleren Entwicklungsprozess.

Nach [Poet09] (S. 174) tragen die objektorientierten Sprachen zwar zu einer besseren Wiederverwendbarkeit bei, sie sind aber keinesfalls als Wundermittel anzusehen. Im Speziellen kann durch die ungeschickte oder fehlerhafte Anwendung der recht mächtigen Sprachkonstrukte nämlich dieser Vorteil schnell verloren gehen. Es ist demnach immer vom Entwurf des Systems abhängig, ob letztendlich eine bessere Wiederverwendbarkeit gegeben ist.

Frederick P. Brooks (* 1931, siehe [@wikio]) setzte in die Objektorientierung bereits im Jahr 1975 in [Broo95] (S. 189 f.) größere Hoffnungen, da deren Konzepte dem Entwickler einen übersichtlichen und eleganten Systementwurf ermöglichten. Er zweifelte jedoch schon damals daran, dass dieses Paradigma zur sogenannten Silberkugel – es handelt sich dabei um einen von Brooks geprägten Begriff für ein Verfahren oder eine Technologie, wodurch der Entwicklungsprozess deutlich beschleunigt bzw. verbessert wird – werden würde. Etliche Jahre später fügte er dann in [Broo95] (S. 219 ff.) ergänzend hinzu, dass das objektorientierte Paradigma zwar Verbesserungen brachte, jedoch eben nicht in den von vielen erhofften Dimensionen.

Einigermaßen kritisch setzt sich [@oszha] mit der Objektorientierung auseinander. Hier werden nicht nur die unterschiedlichen Risiken beim Umstieg auf die objektorientierte

Programmierung behandelt, auch die Probleme in Zusammenhang mit der Wiederverwendung von Komponenten eines vorgegangenen Projekts werden recht umfassend abgehandelt. Demnach ist es Illusion zu glauben, dass diese ohne weiteres Zutun von selbst eintritt. Vielmehr muss bereits beim Erstdesign solcher Komponenten auf eine zukünftige Wiederverwendung geachtet werden. Weiters sind jene zu katalogisieren und müssen den Entwicklern in entsprechender Art und Weise bereitgestellt werden, wobei auch auf die Schulung der Programmierer bzgl. dem Einsatz dieser Komponenten nicht vergessen werden darf.

Zusammenfassend lässt sich festhalten, dass die Objektorientierung heutzutage ein weitverbreitetes und nicht mehr wegzudenkendes Konzept darstellt, wenngleich sie die sehr hoch gegriffenen Erwartungen nicht ganz erfüllen konnte.

3.2 Geschichtliche Entwicklung der Objektorientierung

Die Geschichte der Objektorientierung geht Hand in Hand mit der Entwicklung objektorientierter Programmiersprachen. In Anlehnung an [@wikip] lässt sich die Historie dieser Sprachen wie folgt darstellen:

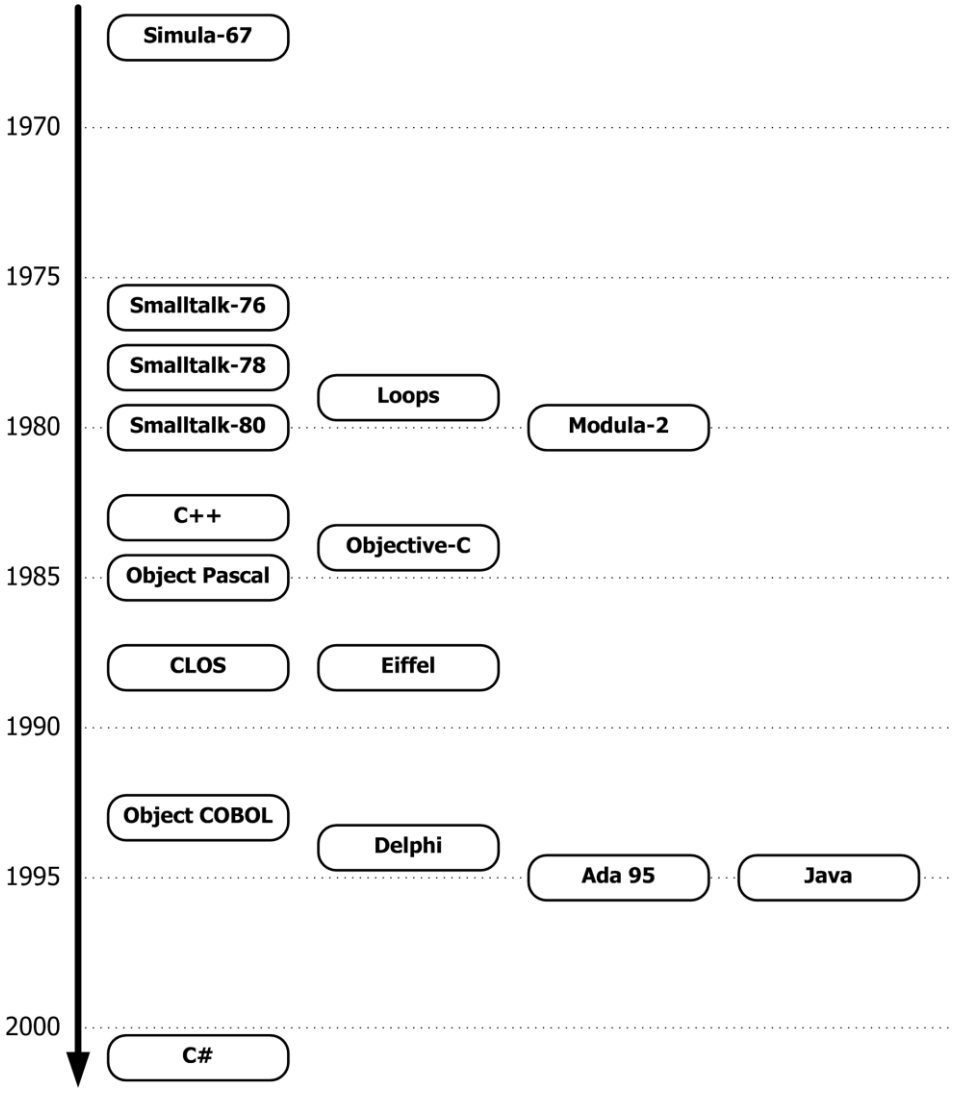


Abbildung 20: Historie objektorientierter Programmiersprachen

Gemäß [Witt92] (S. 1 ff.) gilt Simula-67 als erste objektorientierte Programmiersprache. Sie basierte auf ALGOL-60 und wurde zur computergestützten Simulation von ereignisgesteuerten Systemen entwickelt. Das Besondere an dieser Sprache war, dass sie anstatt der bis dahin üblichen mathematischen Modelle erstmals Objekte zur Repräsentation der zu simulierenden Prozesse verwendete. Jedes Objekt bestand dabei aus lokalen Daten und Prozeduren und konnte mit anderen Objekten kommunizieren. Das Gesamtsystem setzte sich aus einer Menge entsprechend kooperierender Objekte zusammen. Die Zustände und das Verhalten der Objekte wurden über Klassendefinitionen festgelegt. Das Konzept der Vererbung war ebenfalls bereits verfügbar. Trotz dieses revolutionären Ansatzes blieb Simula-67 der Erfolg verwehrt und es fand kaum Verbreitung.

Größeres Interesse an der Objektorientierung kam dann nach [Poet09] (S. 4 f.) erstmals Anfang der 80er Jahre dank Smalltalk auf. Es war dies die erste vollständig objektorientierte Programmiersprache, in der alles in Form von Objekten realisiert war und die über eine große Klassenbibliothek verfügte. Zum Durchbruch reichte es aber trotzdem nicht. Einerseits war die Idee für die damalige Zeit noch zu neu, andererseits hatte es auch mit Performanceproblemen zu kämpfen und verlangte außerdem nach zu viel Speicher.

Richtig populär wurde die objektorientierte Programmierung laut [Bole04] (S. 17) erst Ende der 80er Jahre durch C++. Dabei handelte es sich um eine auf C basierende, allerdings um die objektorientierten Konzepte erweiterte Programmiersprache. Es war eine Zeit, in der die graphischen Benutzeroberflächen immer mehr die textbasierten ablösten. Da sich C++ hervorragend für die Entwicklung solcher Oberflächen eignete, fand es in Entwicklerkreisen eine rasche Verbreitung.

Mitte der 90er Jahre wurde dann Java veröffentlicht. Das Besondere an dieser Sprache war, dass ihr kompakter Bytecode ohne Änderung auf sämtlichen Plattformen lauffähig war, für die eine entsprechende Java Virtual Machine zur Verfügung stand. Wesentlich zum Erfolg von Java trugen auch dessen Konzepte bei, welche die Ausführung von Java-Applikationen im damals immer populärer werdenden Internet ermöglichten.

Kurz nach der Jahrtausendwende kam noch C# auf, das derzeit neben C++ und Java zu den wichtigsten Vertretern der objektorientierten Programmiersprachen zählt. Selbst für Skriptsprachen, wie z. B. Perl, PHP oder Python, gibt es mittlerweile entsprechende objektorientierte Erweiterungen. Der Siegeszug der Objektorientierung ist allerdings schon lange nicht mehr aufzuhalten.

3.3 Objektorientierte Konzepte

Laut [Poet09] (S. 325) ist die wissenschaftliche Diskussion darüber, welches die wesentlichen objektorientierten Konzepte sind, noch nicht abgeschlossen. [Gall95] sieht in Zusammenhang mit der Objektorientierung eine Unmenge an Prinzipien, die sich teilweise überlappen, teilweise aber sogar widersprechen.

Es ist demnach schwierig, die grundlegenden objektorientierten Konzepte eindeutig auszumachen, weshalb mittels folgender Übersicht eine Annäherung an die Thematik erfolgen soll:

	Aufgezählte objektorientierte Konzepte
[Baum96] (S. 278 f.)	• Objekte und Klassen • Klassenhierarchie mit Vererbung
[Poet09] (S. 325 ff.)	• Klassen • Vererbung • Dynamische Methodenauswahl und Subtyping
[Punt08] (S. 10 ff.)	• Objekte • Klassen • Polymorphismus • Vererbung
[Smit93] (S. 7 ff.)	• Unsichtbare Daten (Kapselung) • Hierarchie der Objektdefinitionen • Polymorphie • Einzelner Datentyp
[Witt92] (S. 53 ff.)	• Modularisierung und Datenkapselung • Objektidentität • Nachrichten und Überladen (Polymorphismus) • Klassen, Vererbung und Exemplare

Tabelle 2: Übersicht über objektorientierte Konzepte in Literatur

Die anschließende Einführung in die objektorientierten Konzepte erfolgt nun gemäß [Punt08] (S. 10 ff.).

3.3.1 Objekte

Das Objekt stellt das wichtigste Konzept der objektorientierten Programmierung dar und symbolisiert eine zusammengehörende Einheit in der Programmausführung. Ein nach dem Paradigma der Objektorientierung entwickeltes System setzt sich zur Laufzeit aus einer Menge von Objekten zusammen, die sich zum Teil kennen und untereinander Nachrichten austauschen.

Jedes Objekt kapselt zusammengehörige Variablen und Routinen zu einer Einheit, wobei beide untrennbar miteinander verbunden sind und in einer logischen Beziehung zueinander stehen. Einerseits werden von den Routinen die Variablen zur Ausführung ihrer Aktionen benötigt, andererseits haben diese oftmals nur in Zusammenhang mit einer Routine eine Bedeutung.

Der Zugriff auf ein Objekt erfolgt von außen durch das Zusenden einer Nachricht. Dies kommt dem Aufruf einer öffentlichen Routine des Objektes gleich. Auf die Variablen sollte niemals ein Direktzugriff von außen möglich sein, sondern es sollten entsprechende Routinen dafür vorgesehen werden, sofern dies erwünscht ist.

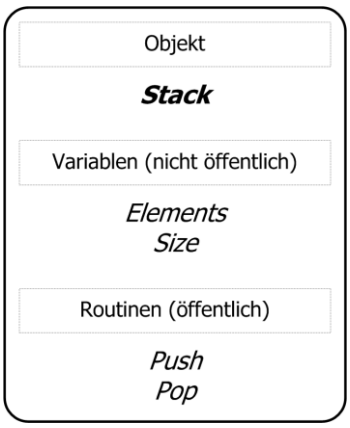


Abbildung 21: Aufbau eines Objektes

Folgende vier Eigenschaften sind charakteristisch für ein Objekt:

- Identität: Jedes Objekt hat eine eindeutige Identität, über die es ansprechbar ist. Dies ermöglicht es, dass einem bestimmten Objekt Nachrichten zugeschickt werden können.
- Zustand: Jedes Objekt besitzt einen Zustand. Dieser entspricht den Werten aller Variablen des Objektes. Durch den Aufruf der öffentlichen Routinen ändert sich im Regelfall der Zustand des zugehörigen Objektes.
- Verhalten: Jedes Objekt weist ein bestimmtes Verhalten auf. Dieses bestimmt, wie sich das Objekt beim Empfang einer Nachricht verhält. Das konkrete Verhalten ist immer von der Nachricht, also dem konkreten Routinenaufruf sowie den übergebenen Parametern und dem aktuellen Zustand des Objektes abhängig.
- Schnittstelle: Jedes Objekt bietet ein oder mehrere Schnittstellen an, die es unterstützt und implementiert. Eine solche Schnittstelle beschreibt das Objektverhalten in dem für den Aufrufer notwendigen Detailliertheitsgrad. Oftmals werden dadurch lediglich die Methodenköpfe der öffentlichen Routinen beschrieben. Die Implementierung legt dann das Verhalten des Objektes im Detail fest. Schnittstellen haben die Aufgabe den Inhalt eines Objektes von dessen Außensicht zu trennen und dadurch für entsprechende Abstraktion zu sorgen.

3.3.2 Klassen

Klassen stellen ein weiteres Konzept der Objektorientierung dar. Jedes Objekt gehört einer bestimmten Klasse an. Diese enthält die Implementierung und beschreibt dadurch das Verhalten der ihr zugehörigen Objekte im Detail.

Jede Klasse beinhaltet außerdem zumindest einen Konstruktor, mittels dem neue Objekte dieser Klasse erzeugt und initialisiert werden können. Man spricht von den Objekten auch als Instanzen dieser Klasse.

Alle Instanzen einer Klasse weisen dieselbe Implementierung und dieselben Schnittstellen auf. Sie haben aber unterschiedliche Identitäten und Variablen, die in diesem Zusammenhang dann als Instanzvariablen bezeichnet werden. Auch die Zustände dieser Objekte weichen meistens voneinander ab.

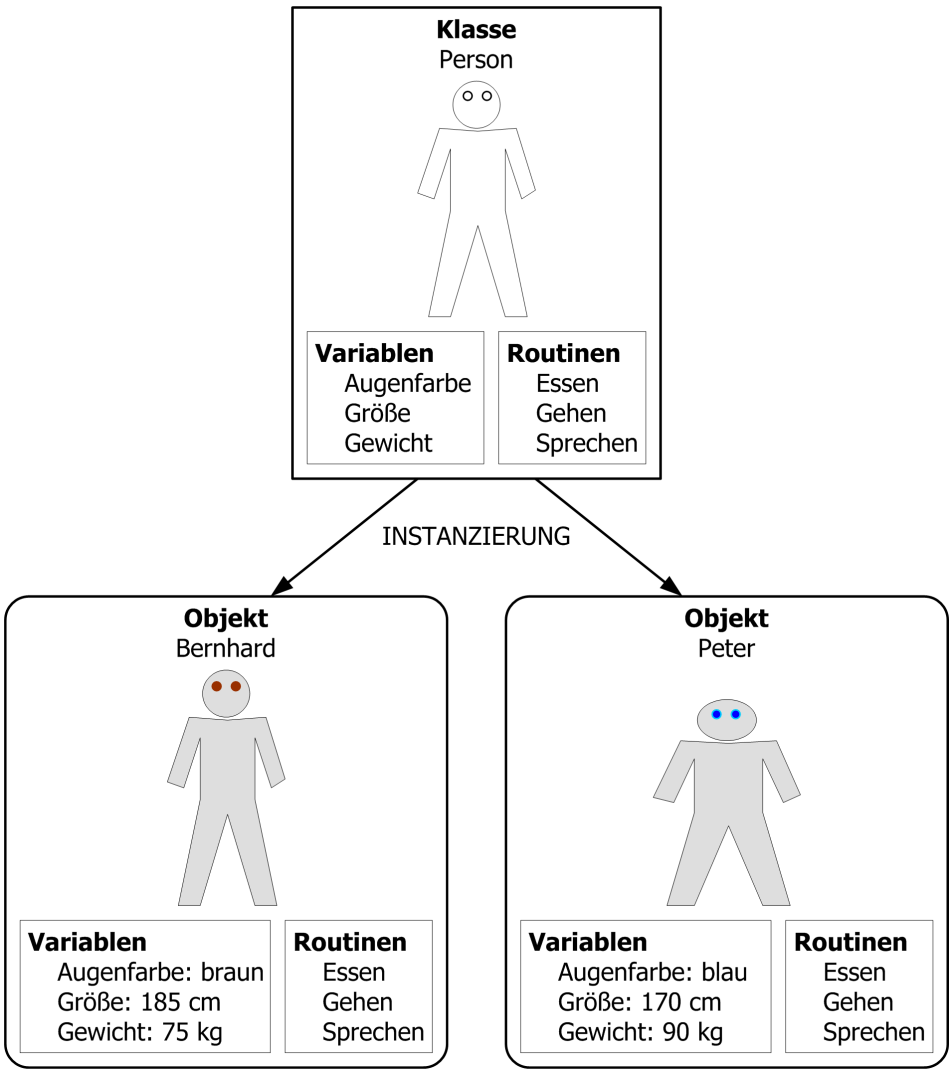


Abbildung 22: Zusammenhang zwischen Klasse und zugehörigen Objekten

Sofern eine Programmiersprache das Klassenkonzept unterstützt, sind von den Entwicklern hauptsächlich Klassen zu implementieren. Die notwendigen Objekte werden dann zur Programmlaufzeit dynamisch über einen Konstruktor der Klasse erzeugt. Objekte können in diesem Fall dann meistens gar nicht ausprogrammiert werden.

3.3.3 Polymorphismus

Der Begriff polymorph entstammt der griechischen Sprache und bedeutet so viel wie vielgestaltig.

In Verbindung mit Programmiersprachen ist dann von Polymorphimus die Rede, wenn eine Variable oder eine Routine gleichzeitig mehrere Typen haben kann. Dies hat zur Folge, dass dem Parameter einer polymorphen Routine Argumente unterschiedlichen Typs übergeben werden können. Objektorientierte Sprachen sind durchgängig polymorph.

Es lassen sich folgende Arten von Polymorphismus unterscheiden, auf die allerdings nicht näher eingegangen werden soll:

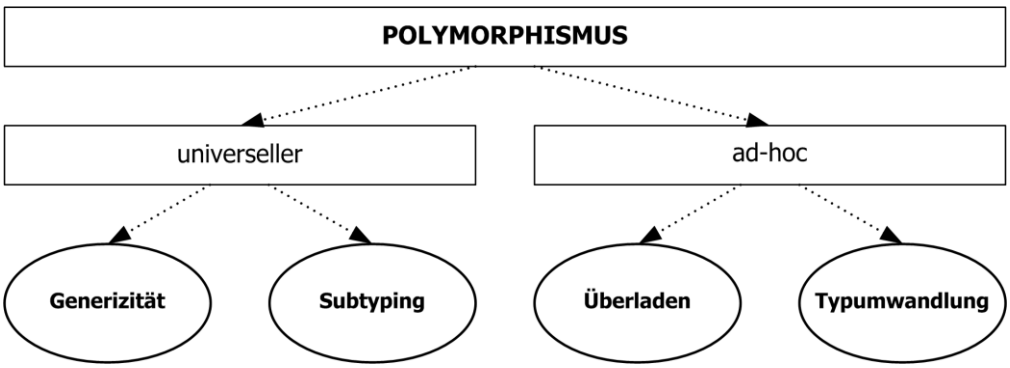


Abbildung 23: Arten von Polymorphismus

3.3.4 Vererbung

Das letzte Konzept der Objektorientierung ist die Vererbung, welche die Ableitung einer neuen Klasse aus einer existierenden ermöglicht. In der neuen, abgeleiteten Klasse – auch Unterklasse genannt – sind nur mehr die Unterschiede zur Basisklasse – auch Oberklasse genannt – anzugeben.

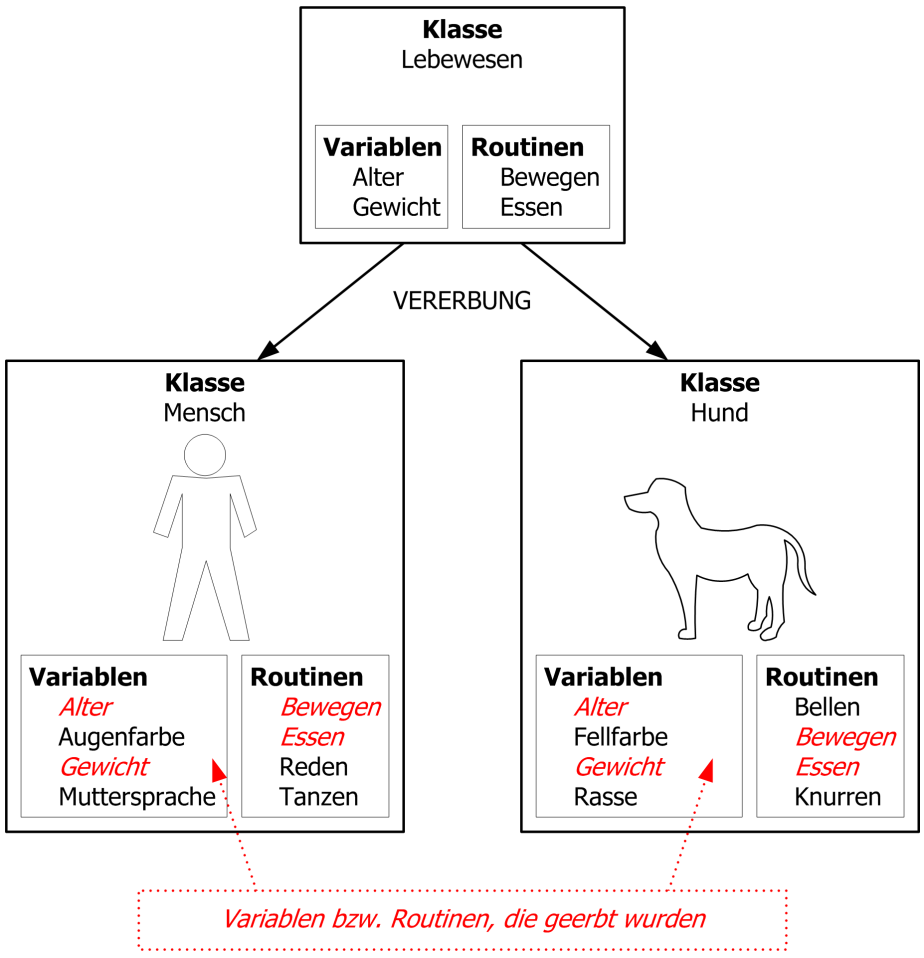


Abbildung 24: Vererbungsprinzip bei Objektorientierung

Der Sinn der Vererbung besteht darin, dem Entwickler Schreibaufwand zu ersparen. Des Weiteren wird durch dieses Konzept die Wartung vereinfacht, denn Änderungen in der Oberklasse werden durch das Vererbungsprinzip automatisch auch an alle Unterklassen weitergegeben.

Zu unterscheiden sind die Einfach- und die Mehrfachvererbung. Bei ersterer wird jede Unterklasse von genau einer Oberklasse abgeleitet, bei zweiterer kann jede Unterklasse auch von mehreren Oberklassen abgeleitet sein. Java unterstützt im Gegensatz zu C++ lediglich die Einfachvererbung, gleicht aber mittels des Interface-Konzepts die daraus hinsichtlich Vererbung resultierenden Nachteile so gut wie aus.

3.4 Objektorientierter Entwicklungsprozess

Der objektorientierte Entwicklungsprozess lässt sich in die Modellierung und die Programmierung aufspalten, wobei die erste Phase üblicherweise wiederum in Analyse und Design unterteilt wird. Der Weg vom Problem zur Lösung lässt sich somit wie folgt darstellen:

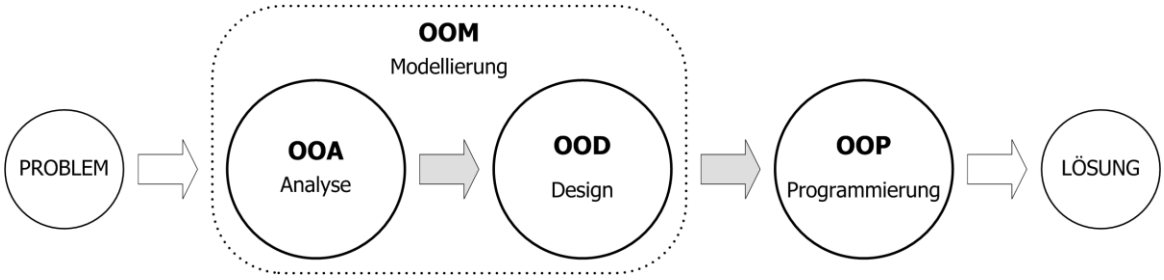


Abbildung 25: Phasen des objektorientierten Entwicklungsprozesses

Gemäß [Hubw07] (S. 105) vermischen sich bei der Objektorientierung oft die Begriffswelten der Modellierung und Programmierung, was eine klare Trennung zwischen diesen beiden Abschnitten erschwert.

3.4.1 Objektorientierte Modellierung (OOM)

Das Ziel der OOM ist es, zu einem Modell der Realität zu gelangen, das dann im Anschluss in ein lauffähiges Programm umgesetzt werden kann.

Laut [@oszhb] ist die Abgrenzung zwischen Analyse und Design nicht immer ganz einfach. Ansatzweise lassen sich diese beiden Phasen, die im Anschluss gemäß dem vorhin genannten Werk genauer beschrieben werden sollen, folgendermaßen charakterisieren:

- Die Analyse legt fest, was das System tun soll.
- Das Design legt fest, wie das System dies tun wird.

Als De-facto-Standard für die objektorientierte Modellierung hat sich in den letzten Jahren die Unified Modeling Language (UML) etabliert.

3.4.1.1 Objektorientierte Analyse (OOA)

Im Rahmen der OOA sollen die notwendigen Objekte gefunden und arrangiert werden, die durch entsprechende Kooperation das reale System, auch als Fachkonzept bezeichnet, abbilden.

Zuerst gilt es, die Objekte bzw. Klassen auszumachen, danach deren Attribute und Methoden zu bestimmen und zu guter Letzt die Beziehungen zwischen den einzelnen Klassen festzulegen.

Es soll an dieser Stelle darauf geachtet werden, dass die Geschäftslogik von der Benutzeroberfläche getrennt wird, damit zweitere im Bedarfsfall problemlos ausgetauscht werden kann. Weiters darf das am Ende dieser Phase vorliegende Modell weder eine technische Lösung, noch irgendwelche Optimierungen hinsichtlich der Zielsprache enthalten.

3.4.1.2 Objektorientiertes Design (OOD)

In der Phase des OOD wird nun die endgültige Architektur des zu erstellenden Systems bestimmt.

Es wird dabei festgelegt, wie die Fachklassen an die Benutzeroberfläche angebunden werden, auf welche Art die Daten persistent gemacht werden, welche Klassenbibliotheken verwendet werden, wie die Anbindung an bestimmte Schnittstellen erfolgt und welche Optimierungen in Bezug auf die Zielsprache durchgeführt werden.

3.4.2 Objektorientierte Programmierung (OOP)

Im letzten Schritt des objektorientierten Entwicklungsprozesses gilt es gemäß [@learb] unter Verwendung einer geeigneten Programmiersprache aus dem Modell ein lauffähiges Programm zu erstellen.

Nach Abschluss der OOP sollte ein ausführbares Programm vorliegen, das eine entsprechende softwaretechnische Abstraktion des ursprünglichen, realen Problems darstellt.

3.5 Objektorientierte Programmiersprachen

Eine Übersicht über die wichtigsten Vertreter der objektorientierten Programmiersprachen findet sich unter [@wikiq]. Da eine Beschäftigung mit mehreren dieser Sprachen den Rahmen dieser Arbeit sprengen würde, beschränken sich die anschließenden Betrachtungen auf Java.

3.5.1 Java

Die Programmiersprache Java unterstützt wie alle modernen objektorientierten Programmiersprachen sämtliche oben genannte Konzepte. Wie diese konkret in dieser Sprache umgesetzt sind, wird in Folge unter Verwendung des Beispiels aus der Abbildung 24 (S. 46) demonstriert.

Es muss an dieser Stelle noch erwähnt werden, dass in dieser Arbeit keine umfassende Einführung in Java erfolgen kann – auch das würde weit über das eigentliche Ziel dieser Arbeit hinausschießen –, sondern es soll lediglich gezeigt werden, wie man sich die konzeptionelle Umsetzung der erwähnten objektorientierten Konzepte in Java vorzustellen hat.

3.5.1.1 Objekte

Objekte können in Java nicht ausprogrammiert werden, die Arbeit des Entwicklers beschränkt sich auf die Implementierung von Klassen. Durch entsprechende Konstruktoren kann er dann zur Laufzeit dynamisch Instanzen einer Klasse erzeugen. Um das Löschen eines Objektes muss sich der Programmierer nicht kümmern, denn Java gibt automatisch den Speicher von nicht mehr referenzierten Objekten frei. Dieses Prinzip nennt sich Garbage Collection.

Um ein Objekt namens meinLebewesen der Klasse Lebewesen zu erzeugen, ist folgender Code zu schreiben:

```
Lebewesen meinLebewesen = new Lebewesen();
```

Bei Vorhandensein eines erweiterten Konstruktors, können bei der Instanzierung auch gleich die zu setzenden Attributswerte für das Alter und das Gewicht übergeben werden:

```
Lebewesen meinLebewesen = new Lebewesen(10, 50);
```

Zum Setzen der Attributswerte nach der Instanzierung des Objektes, bedient man sich entsprechender Getter- und Setter-Methoden. Dank dieser kann der direkte Zugriff von außen auf die internen Variablen des Objektes untersagt werden und der Entwickler hat trotzdem noch die Möglichkeit, diese abzufragen bzw. zu verändern.

Um das aktuelle Alter des Objektes abzufragen, in einer temporären Variable zwischenzuspeichern und anschließend um 10 erhöht wieder im Objekt zu setzen, muss der Entwickler folgendes schreiben:

```
int temp = meinLebewesen.getAlter();
meinLebewesen.setAlter(temp + 10);
```

Zu guter Letzt gilt es noch zu klären, wie Routinen eines Objektes aufgerufen werden können. Im Grunde wurde dies soeben anhand der Getter- und Setter-Methoden gezeigt. Für jede andere Methode hat dies analog zu erfolgen.

Um ein Lebewesen durch den Aufruf der Methode bewegen virtuell in Bewegung zu setzen, muss der Entwickler folgenden Quellcode schreiben:

```
meinLebewesen.bewegen();
```

3.5.1.2 Klassen

Wie bereits vorhin festgehalten wurde, implementiert der Entwickler keine Objekte, sondern schreibt Klassen mit entsprechender Funktionalität.

Für das Lebewesen sieht der Quelltext der zugehörigen Klasse wie folgt aus:

```
public class Lebewesen {
  int alter;
  int gewicht;
```

```
  public Lebewesen() {
    // Standardwerte setzen
    alter = 0;
    gewicht = 75;
  }

  public Lebewesen(int alter,
                   int gewicht) {
    this.alter = alter;
    this.gewicht = gewicht;
  }

  public int getAlter() {
    return alter;
  }
  public void setAlter(int alter) {
    this.alter = alter;
  }

  public int getGewicht() {
    return gewicht;
  }
  public void setGewicht(int gewicht) {
    this.gewicht = gewicht;
  }

  public void bewegen() {
    // Aktionen ausführen ...
  }

  public void essen(String lebensmittel) {
    // Aktionen ausführen ...
  }

  public void essen(int kalorien) {
    // Aktionen ausführen ...
  }
}
```

Am Beginn der Klassendefinition wird nach dem Schlüsselwort class der gewünschte Name der Klasse angeführt:

```
public class Lebewesen
```

Danach werden üblicherweise die internen Variablen der Klasse definiert, die im Falle der Klasse Lebewesen wie folgt lauten:

```
int alter;
int gewicht;
```

Es folgen dann im Anschluss die Konstruktoren, mittels denen man zur Laufzeit neue Objekte dieser Klasse erzeugen kann. Die Klasse Lebewesen hat hier zwei Konstruktoren:

```
public Lebewesen()
```

```
public Lebewesen(int alter,
                 int gewicht)
```

Um den Zugriff von außen auf die Variablen zu ermöglichen, werden jede Menge Getter- und Setter-Methoden definiert, auf die nicht näher eingegangen werden soll.

Zu guter Letzt erfolgt die Definition der eigentlichen Klassenmethoden. Bei diesem Beispiel werden zusätzlich zu den Getter- und Setter-Methoden noch drei Routinen implementiert, deren Methodenköpfe folgendes Aussehen haben:

```
public void bewegen()
public void essen(String lebensmittel)
public void essen(int kalorien)
```

3.5.1.3 Polymorphismus

Wie bereits bei den Konzepten beschrieben wurde, steht polymorph für vielgestaltig und bewirkt bei objektorientierten Sprachen, dass in Abhängigkeit des bzw. der übergebenen Argumenttypen die jeweils zugehörige Methode aufgerufen wird.

Die Methode essen eines Lebewesens kann man hier entweder mit einem String- oder einem int-Parameter aufrufen:

```
meinLebewesen.essen("Joghurt");
meinLebewesen.essen(500);
```

Je nachdem wird dann vom System zur Laufzeit eine andere Klassenmethode aufgerufen und ausgeführt. Das Objekt zeigt hier also zur Laufzeit in Bezug auf die Methode essen ein polymorphes Verhalten.

3.5.1.4 Vererbung

Die Vererbung wird anhand der Klassen Lebewesen und Mensch demonstriert. In Java ist es immer vom Sichtbarkeitsbereich einer Variable bzw. einer Methode abhängig, in welcher Form in der Unterklasse darauf zugegriffen werden kann bzw. ob ein Überschreiben möglich ist.

Für die vom Lebewesen abgeleitete Klasse Mensch sieht der Quelltext wie folgt aus:

```
public class Mensch extends Lebewesen {
  String augenfarbe;
  String muttersprache;

  public Mensch() {
    // Standardwerte setzen
    alter = 0;
    augenfarbe = "braun";
    gewicht = 75;
    muttersprache = "deutsch";
  }
  public Mensch(int alter,
                String augenfarbe,
                int gewicht,
```

```
                String muttersprache) {
    super(alter, gewicht);
    this.augenfarbe = augenfarbe;
    this.muttersprache = muttersprache;
  }

  public String getAugenfarbe() {
    return augenfarbe;
  }
  public void setAugenfarbe(String augenfarbe) {
    this.augenfarbe = augenfarbe;
  }
  public String getMuttersprache() {
    return muttersprache;
  }
  public void setMuttersprache(String muttersprache) {
    this.muttersprache = muttersprache;
  }

  public void reden() {
    // Aktionen ausführen ...
  }

  public void tanzen() {
    bewegen();
    // Aktionen ausführen ...
  }
}
```

Das Schlüsselwort für die Vererbung heißt extends und folgt inklusive dem Namen der Oberklasse der einleitenden Klassendefinition:

```
public class Mensch extends Lebewesen
```

Auf die geerbten Variablen Alter und Gewicht aus der Oberklasse Lebewesen kann in der Unterklasse Mensch genauso wie auf lokale Variablen zugegriffen werden, wie die folgenden beiden Zeilen in der Klassendefinition zeigen:

```
alter = 0;
gewicht = 75;
```

Selbst der Konstruktor der Oberklasse kann mittels des Schlüsselwortes super aus der Unterklasse heraus aufgerufen werden:

```
super(alter, gewicht);
```

Die in der Oberklasse Lebewesen definierten Methoden können sowohl innerhalb der Klassendefinition von Mensch – dies zeigt sich in der Methode tanzen, die sich der Methode bewegen bedient – als auch in Zusammenhang mit einer Objektreferenz von außen aufgerufen werden, wie das folgende kurze Codebeispiel zeigt:

```
Mensch meinMensch = new Mensch();
meinMensch.bewegen();
```

4 LEGO Mindstorms

Dieses Kapitel beginnt mit einer Einführung in LEGO Mindstorms und informiert anschließend über dessen geschichtliche Entwicklung. Danach erfolgt anhand der NXT-Version eine Auseinandersetzung mit dem Aufbau und der Programmierung. Abgerundet wird der Abschnitt mit der Vorstellung etlicher ausgewählter Projekte, die im Ausbildungssektor angesiedelt sind.

4.1 Einführung

Laut dem Hersteller blickt LEGO Mindstorms mittlerweile auf viele erfolgreiche Jahre zurück. Nach einer allgemeinen Einführung zum Begriff des Roboters sollen deshalb neben LEGO Mindstorms auch die Erfolgsfaktoren dafür betrachtet werden. Zu guter Letzt wird noch ein Blick auf alternative Robotersysteme geworfen.

4.1.1 Roboter

Die Beschäftigung mit LEGO Mindstorms bedingt zwangsläufig, sich mit dem Begriff des Roboters auseinanderzusetzen. [@wikt] versteht darunter eine Maschine, die sich eigenständig um die Durchführung einer bestimmten Aufgabe kümmert.

Grundsätzlich stellt ein Roboter ein eingebettetes System dar und besteht aus Hardware- und Softwareteilen. Das Herzstück eines jeden Roboters ist ein entsprechender Controller, der das Programm ausführt und in Abhängigkeit davon die Aktoren und Sensoren ansteuert, um Aktionen durchzuführen bzw. um Messwerte abzufragen. Dazu kommen noch die Firmware, die grundlegende Funktionalitäten zur Verfügung stellt und meist ebenfalls austauschbar ist sowie die restlichen mechanischen Bauteile. Dieser Sachverhalt lässt sich wie folgt zusammenfassen:

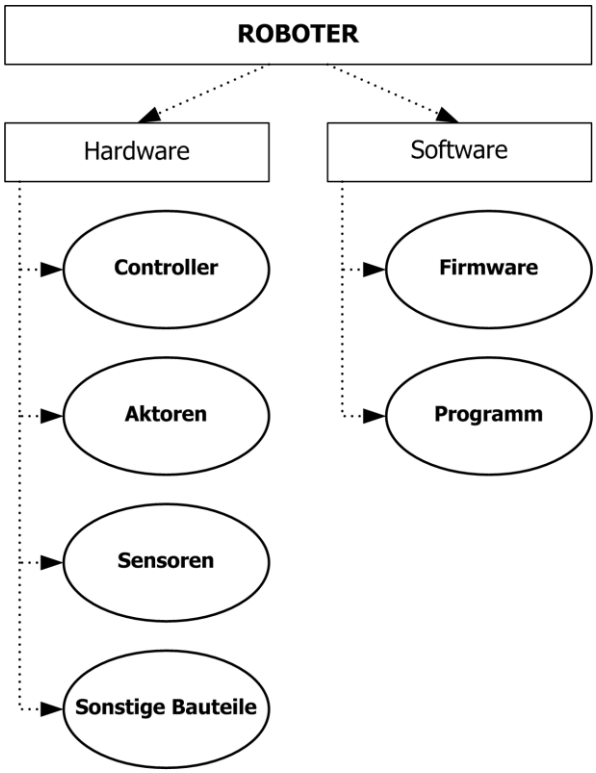


Abbildung 26: Wichtigste Bestandteile eines Roboters

Vergleicht man weiters den Aufbau eines Roboters mit dem eines Menschen, so erkennt man schnell, wer jenem als Vorbild dient:

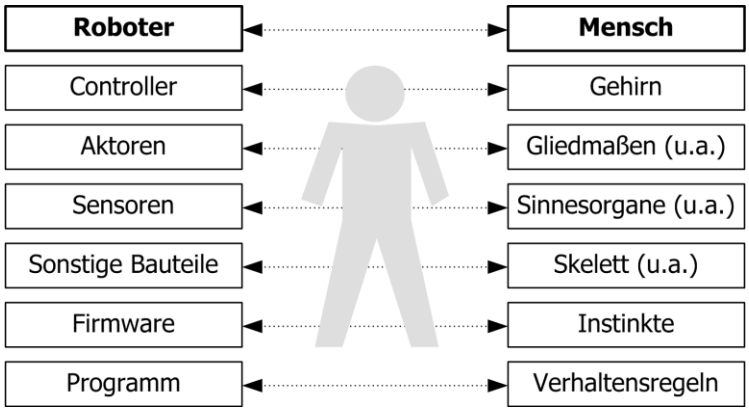


Abbildung 27: Vergleich zwischen Aufbau eines Roboters und dem Menschen

Bei der Konstruktion eines Roboters ist gemäß [Bagn07] (S. 139 ff.) zu beachten, dass der primäre Zweck des Roboters stets im Auge behalten wird und sich die Entwicklung ganz danach ausrichtet. Nach der Funktionsbestimmung wird dann beim Bau mit den wichtigsten Teilen begonnen und diese werden dann schrittweise ergänzt. Des Weiteren werden elf Eigenschaften genannt, die beim Bau eines jeglichen Systems von Bedeutung sind:

- Geschwindigkeit
- Belastbarkeit
- Leistung
- Agilität
- Stabilität
- Genauigkeit
- Symmetrie
- Kompaktheit
- Minimalismus
- Robustheit
- Modularität

4.1.2 LEGO Mindstorms

Gemäß [@wikir] steht LEGO Mindstorms für eine Produktreihe der dänischen Firma LEGO. Es sind unterschiedliche Baukästen dieser Serie erhältlich, die neben Unmengen von LEGO Technic-Elementen jeweils einen programmierbaren Baustein sowie unterschiedliche Aktoren und Sensoren enthalten. Mittels LEGO Mindstorms können verschiedene Roboter programmiert werden, wobei sich diese je nach Bauart in eine der folgenden vier Kategorien einteilen lassen:

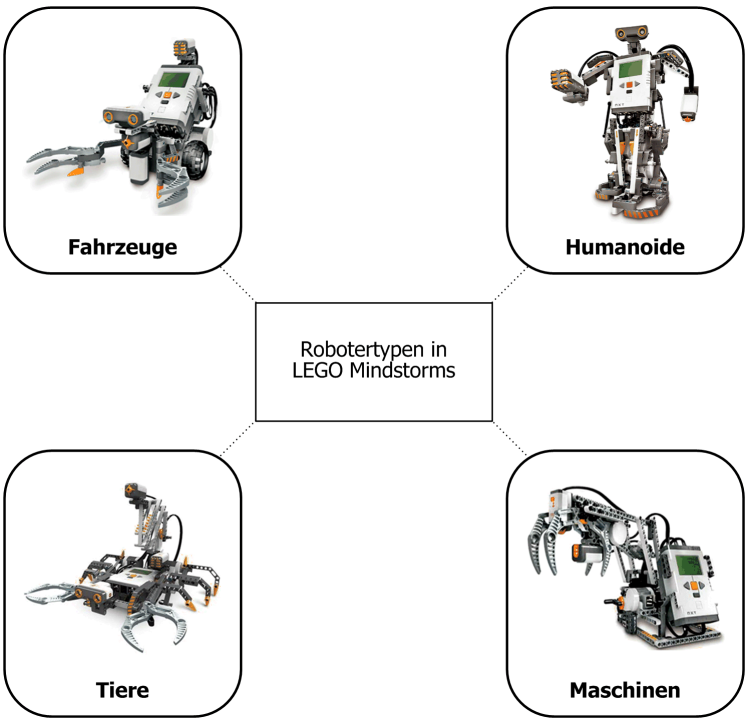


Abbildung 28: Robotertypen in LEGO Mindstorms

Zur Faszination an LEGO Mindstorms trägt vor allem die Vielzahl an Möglichkeiten der Verwendung bei. So lassen sich dank der Unmengen an beiliegenden Bauteilen nicht nur die phantasievollsten Gebilde zusammenbauen, sondern ihnen kann dank der vielfältigen Programmiermöglichkeiten auch unterschiedlichstes Verhalten eingeimpft werden. Hat man sich mit einem seiner Kunstwerke zur Genüge beschäftigt, dann kann der Roboter problemlos zerlegt und einfach ein neuer konstruiert und programmiert werden. Zusätzlich sind auch Erweiterungssets und Sensoren von Fremdanbietern erhältlich, welche die Grenzen des Möglichen weiter ausdehnen.

LEGO Mindstorms ist nach [Bagn07] (S. 2) aber nicht nur ein Spielzeug, sondern hat auch für das Ingenieurwesen größere Bedeutung. Und zwar gelangen solche Baukästen bei der Entwicklung von Prototypen, die zu Testzwecken rasch zur Verfügung stehen sollen, immer wieder zum Einsatz. Des Weiteren wird LEGO Mindstorms im Ausbildungsbereich, speziell im offenen Schulunterricht, eingesetzt. Auch alle möglichen Wettbewerbe haben sich über die Jahre etabliert und motivieren nicht nur Kinder, sondern auch Erwachsene zur intensiven Auseinandersetzung mit den zugehörigen Bausätzen.

4.1.3 Alternativsysteme

LEGO Mindstorms nimmt zwar bzgl. Roboterbaukästen die Vormachtstellung am Markt ein, ganz ohne Konkurrenz ist es aber dennoch nicht. [Bagn07] (S. 19 ff.) nennt als Alternativen zu LEGO Mindstorms folgende drei Systeme:

- VEX kit: Dieses System verfügt zwar über wesentlich mehr Eingangs- und Ausgangsports, allerdings sind die erhältlichen Sensoren bei Weitem nicht so leistungsfähig wie bei LEGO Mindstorms. Dazu kommt, dass die meisten Bauteile aus dem nicht besonders anwenderfreundlichen Material Metall bestehen. Es ist etwas komplexer als LEGO Mindstorms, dafür aber weniger raffiniert.

- fischertechnik ROBO Mobile Set: An dieses System lassen sich vierzehn Sensoren anschließen, wodurch es diesbezüglich LEGO Mindstorms ebenfalls deutlich übertrumpft. Die unterschiedlichen Baukästen zielen eher auf das fortgeschrittenere Publikum ab, wobei die Vermittlung der mechanischen Konzepte im Vordergrund steht.
- Handyboard: Zum Zusammenbauen sind elektronische Kenntnisse vonnöten – es muss gelötet werden –, wodurch sich die Zielgruppe deutlich beschränkt. Im Gegensatz zu den anderen genannten wirkt es außerdem mittlerweile etwas in die Jahre gekommen.

4.2 Geschichtliche Entwicklung von LEGO Mindstorms

Nach [Altm05] (S. 5 ff.) ist die Entwicklungsgeschichte von LEGO Mindstorms eng mit dem Namen Seymour Papert (* 1928, siehe [@wikis]) verbunden. Der Südafrikaner beschäftigte sich in den späten 60er Jahren im Rahmen seiner Forschungstätigkeit damit, wie man Kindern auf einfache und interessante Art und Weise die Programmierung von Computern näherbringen könnte.

Um diesen den Einstieg zu vereinfachen, entwickelte er die Programmiersprache Logo, die im Gegensatz zu den damals gängigen Sprachen nicht auf mathematischen, sondern auf logischen Beschreibungen aufbaute. Um das Interesse der Kinder am Programmieren zu wecken, sollte Logo zur Steuerung einfacher Roboter eingesetzt werden.

Über die Jahre entwickelte sich nicht nur die Idee, sondern auch die Technik entsprechend weiter und Mitte der 80er Jahre war es dann so weit, dass man mittels Logo über eine zentrale Schnittstelle auf Aktoren und Sensoren zugreifen konnte. In diesen Jahren begann dann auch die Zusammenarbeit mit LEGO und schon bald wurde ein Robotersystem namens LEGO TC Logo vorgestellt. Für den Bau der Roboter wurden die damals neuen LEGO Technic-Elemente verwendet, die Steuerung erfolgte mittels Logo. Ein großer Nachteil war allerdings, dass die zentrale Schnittstelle fest mit dem Rechner verbunden sein musste, wodurch man keine mobilen Roboter bauen konnte.

So entstand die Idee, die Steuereinheit derart umzugestalten, dass diese vom Roboter mitgetragen werden konnte. Der erste Prototyp eines solchen programmierbaren Bausteins erwies sich aber als nicht zuverlässig genug, die darauffolgende Generation geriet etwas zu groß und schwer. Man lernte allerdings aus den Fehlern und 1998 war es dann soweit: Mit dem Robotics Invention System (RIS) kam der erste Baukasten der Produktreihe LEGO Mindstorms auf den Markt. Das Herzstück war dabei der programmierbare RCX-Stein, der mittels einer graphischen Entwicklungsumgebung programmiert werden konnte. Im originalen Bausatz waren weiters zwei Motoren, zwei Berührungssensoren und ein Lichtsensor enthalten.

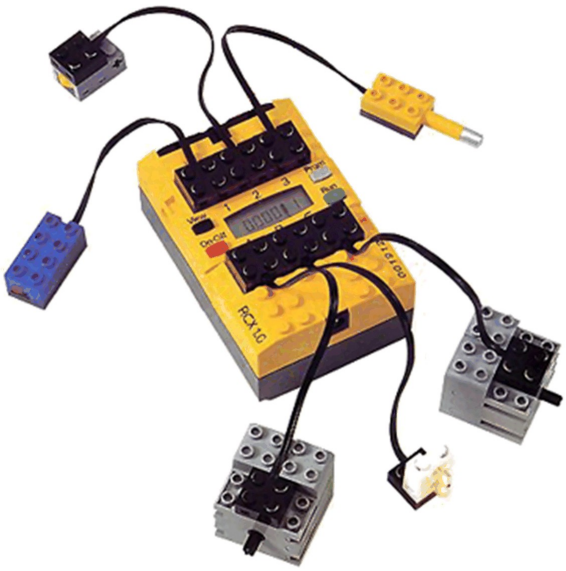


Abbildung 29: RCX-Stein mit Aktoren und Sensoren

Binnen kürzester Zeit eroberte dieses Robotersystem den Markt. Gemäß [Bagn07] (S. 2 ff.) setzten nicht nur viele Schulen diesen Baukasten als unterstützenden Lernbehelf ein, sondern hielt er auch an Universitäten Einzug und war plötzlich Gegenstand diverser Forschungsprojekte. Überraschenderweise fand aber auch die Hackerszene Gefallen an LEGO Mindstorms. Schon recht bald waren die Geheimnisse um den RCX-Stein geknackt und dadurch konnten alternative Sprachen für dessen Programmierung entwickelt und verwendet werden. Wie sich später herausstellte, trug diese Sprachenvielfalt massiv zur Verbreitung des Baukastens bei.

Nachdem man seitens LEGO festgestellt hatte, dass rund 70 % der Benutzer dem Erwachsenenbereich angehörten, brachte man technisch einfachere und billigere Varianten für die eigentliche Zielgruppe, nämlich die Kinder, auf den Markt, die unter den Namen Scout, Spybotics und Cybermaster vertrieben wurden.

Nach vielen Jahren des Wartens wurde dann im Jahr 2006 der Nachfolger der RCX-Version namens LEGO Mindstorms NXT vorgestellt. Dank des deutlich verbesserten, programmierbaren NXT-Steins und etlicher neuer Sensoren wurde dem Benutzer mit diesem Bausatz ein noch leistungsfähigeres Robotersystem zur Verfügung gestellt. Und die fleißige Community sorgte dafür, dass auch für dieses Produkt bald unterschiedliche Programmiersprachen und -umgebungen verfügbar waren.

Abbildung 30: NXT-Stein mit Aktoren und Sensoren

Anfang 2009 kam dann LEGO Education WeDo auf den Markt. Dieses Robotersystem zielt auf sieben- bis elfjährige Kinder ab und soll sie auf sehr spielerische Art und Weise in die Robotik einführen. Im August des gleichen Jahres wurde außerdem LEGO Mindstorms NXT 2.0 veröffentlicht, das gegenüber der Erstversion um einen Sensor erweitert wurde, der Farben erkennen kann.

Bemerkenswert an der Geschichte von LEGO Mindstorms ist jedenfalls, dass ein Spielzeug eine solch hohe Popularität und Verbreitung, und zwar zum Großteil im Erwachsenenbereich erreichen konnte.

4.3 Aufbau von LEGO Mindstorms NXT

LEGO Mindstorms NXT folgt wenig überraschend dem oben beschriebenen Grundaufbau eines Roboters. Im Mittelpunkt steht der programmierbare NXT-Stein, der bis zu drei Servomotoren ansteuern und zur Erfassung der Umgebung auf einen Berührungs-, einen Geräusch-, einen Licht- und einen Ultraschallsensor zurückgreifen kann. Für den Zusammenbau des Roboters stehen dem Konstrukteur weiters über fünfhundert LEGO Technic-Elemente zur Verfügung.

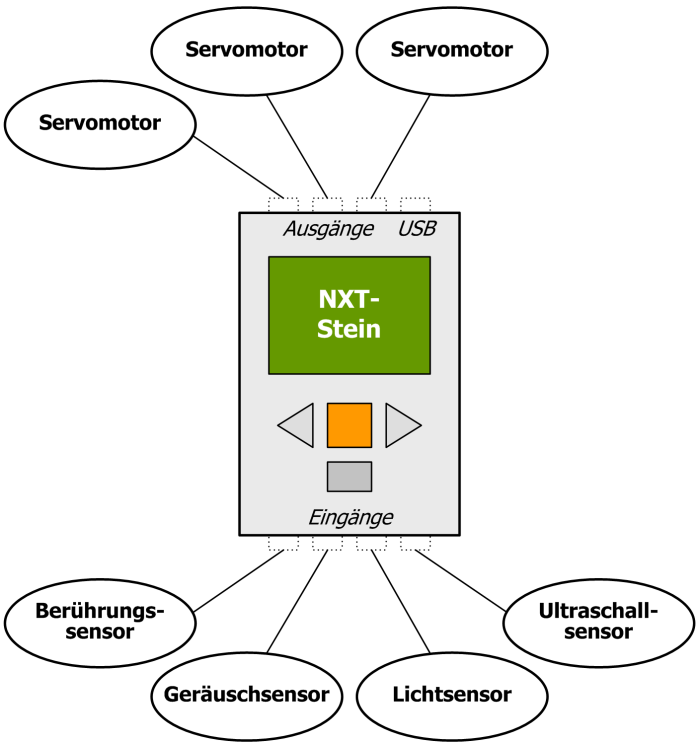


Abbildung 31: Wichtigste Bestandteile von LEGO Mindstorms NXT

Diese Komponenten werden nun in Anlehnung an [Schm09] (S. 8 ff.) bzw. [Bagn07] (S. 4 ff.) näher vorgestellt. Erwähnt werden sollte außerdem noch, dass zusätzlich zu den im NXT-Baukasten enthaltenen Sensoren weitere am Markt erhältlich sind. Eine Beschreibung dieser findet sich unter [Asto07] (S. 88 ff.).

4.3.1 NXT-Stein

Abbildung 32: NXT-Stein

Der programmierbare NXT-Stein stellt das Herzstück eines jeden LEGO Mindstorms-Roboters dar. Er besitzt einen 32-Bit-Mikroprozessor, der mit 48 MHz Taktfrequenz betrieben wird und auf 256 KB Flash- sowie 64 KB Arbeitsspeicher zugreifen kann. In Sachen Stromversorgung benötigt er sechs AA-Batterien oder eine entsprechend kompatible Lithium-Ionen-Batterie.

Für den Anschluss von Servomotoren stehen drei Ausgangsports zur Verfügung. Um einen Sensor anzuschließen, ist dieser mittels Kabel an einen der vier Eingangsports

anzustecken. Bei Bedarf können über einen entsprechenden Adapter auch mehr als vier Sensoren gleichzeitig angeschlossen werden. Zur Kommunikation zwischen dem NXT-Stein und den angehängten Sensoren wird das Protokoll Inter-Integrated Circuit (I^2C) verwendet.

Am NXT-Stein findet sich weiters ein monochromes Liquid Crystal Display (LCD) mit einer Auflösung von 100 x 64 Pixel. Darüber wird einerseits das Menü dargestellt – die Navigation erfolgt hierbei über vier Buttons, deren Verhalten bei Bedarf ebenfalls individuell programmierbar ist –, andererseits lassen sich auf diesem auch eigene Texte, Grafiken und Animationen ausgeben. Auch ein Lautsprecher ist integriert, mittels dem Geräusche und Töne abgespielt werden können.

Zum Transfer des Programms vom Rechner auf den NXT-Stein kann wahlweise Universal Serial Bus (USB) oder Bluetooth verwendet werden. Erstere Übertragungsart hat dabei den Vorteil, dass sie sich auch zum Einspielen einer neuen Version der Firmware eignet. Mittels Bluetooth lassen sich dafür die Daten drahtlos auf den NXT-Stein übertragen, wodurch man sich speziell in der Programmier- und Testphase das lästige Ein- und Ausstecken des Kabels erspart.

4.3.2 Servomotor

Abbildung 33: Servomotor

Im NXT-Baukasten sind drei Servomotoren enthalten. Dank des Getriebes kann bei deren Betrieb zwischen Geschwindigkeit und Leistung gewählt werden. Des Weiteren sind in den Servomotoren Drehzahlmesser eingebaut, die jederzeit über den aktuellen Rotationsstand der Achse Bescheid wissen und dadurch eine Vielzahl an Möglichkeiten eröffnen. So kann jeder Motor nicht einfach nur manuell gestartet und manuell gestoppt werden, sondern es ist auch möglich, jene durch Vorgabe von Umdrehungsgraden bzw. Umdrehungsanzahlen entsprechend zu steuern. Auch die Synchronisation mehrerer Motoren wird dank des integrierten Tachometers ermöglicht.

4.3.3 Berührungssensor

Abbildung 34: Berührungssensor

Dieser Sensor reagiert auf Druckkontakt und unterscheidet lediglich zwei Zustände: Und zwar liefert er zurück, ob der auf dem Sensor angebrachte Button gedrückt ist oder nicht.

4.3.4 Geräuschsensor

Abbildung 35: Geräuschsensor

Dieser Sensor misst den Schalldruckpegel. Der gemessene Wert kann dabei nicht nur in Dezibel zurückgeliefert werden, sondern es ist auch möglich, diesen in Abhängigkeit seiner Frequenz bewerten zu lassen, so wie dies beim menschlichen Gehör der Fall ist. Man spricht in diesem Fall vom A-bewerteten Schalldruckpegel, dessen Einheit in dB(A) angegeben wird.

4.3.5 Lichtsensor

Abbildung 36: Lichtsensor

Dieser Sensor misst die Intensität des in die Linse eintretenden Lichtes. Zum Aufhellen der Szene lässt sich bei Bedarf eine rote Leuchtdiode dazuschalten. Für das menschliche Auge unsichtbares Licht, wie z. B. Infrarotlicht, erkennt dieser Sensor ebenfalls. Er kann nicht nur zur Unterscheidung zwischen hellen und dunklen Objekten genutzt werden, sondern sogar zur Farbbestimmung.

4.3.6 Ultraschallsensor

Abbildung 37: Ultraschallsensor

Dieser Sensor sendet Schallwellen aus, die im Ultraschallbereich angesiedelt sind und dadurch vom menschlichen Gehör nicht wahrgenommen werden können. Durch Messung der Zeitspanne, wie lange es bis zum Eintreffen der reflektierten Wellen beim Sensor dauert, bestimmt der Ultraschallsensor schließlich die Entfernung zu einem Hindernis und liefert jene in Zentimeter zurück.

4.4 Programmierung von LEGO Mindstorms NXT

Damit sich ein Roboter eigenständig um die Durchführung einer bestimmten Aufgabe kümmern kann, muss er entsprechend programmiert werden. Zuerst soll der allgemeine Entwicklungsprozess und anschließend sollen etliche Programmiersprachen und -umgebungen für LEGO Mindstorms NXT betrachtet werden.

4.4.1 Entwicklungsprozess

Die Entwicklung eines Roboters lässt sich in Anlehnung an [Dagd05] am besten mittels eines iterativen Prozesses bewältigen, der wie folgt aussieht:

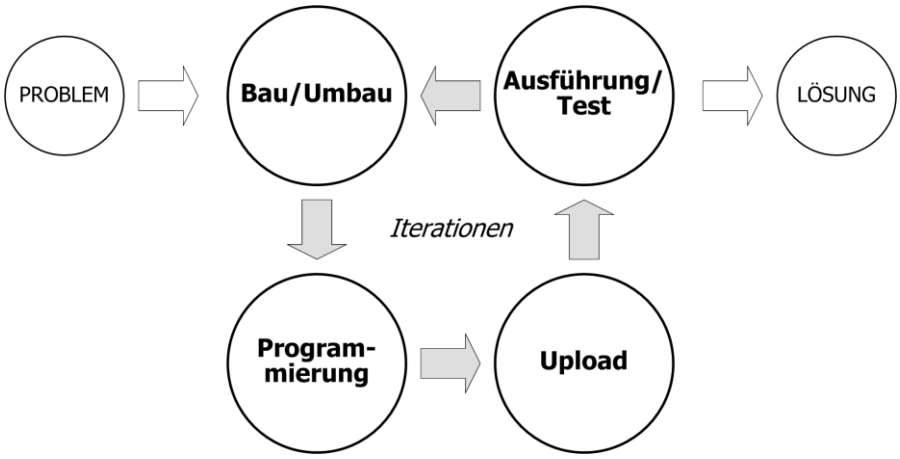


Abbildung 38: Phasen des iterativen Entwicklungsprozesses

Zuerst wird der Roboter gebaut. Danach wird das Programm erstellt und auf den Roboter übertragen. In der letzten Phase der Iteration wird dieses dann vom Roboter ausgeführt, wodurch sein Verhalten getestet werden kann. Je nachdem beginnt dann der Prozess von neuem – in den meisten Fällen wird man den Roboter zwecks Korrektur bzw. Optimierung umbauen bzw. umprogrammieren – oder aber die gewünschte Lösung liegt vor, wodurch der Prozess abgeschlossen ist.

4.4.2 Programmiersprachen und -umgebungen

Für die vorhin beschriebenen Phasen der Programmierung und des Programmuploads stehen unterschiedliche Programmiersprachen und -umgebungen zur Verfügung. [@wikir] und [Hirs03] geben einen guten Überblick über die Entwicklungswerkzeuge, wobei zu beachten ist, dass viele der genannten lediglich mit der RCX-Variante funktionieren.

Grundsätzlich hat man die Wahl zwischen graphischen und textbasierten Programmierumgebungen. Gemäß [Hirs03] ist die Auswahl letztendlich immer von der Antwort auf die Frage "Was soll wem gelehrt werden, und warum?" abhängig. Hinsichtlich des Lehrstoffes lässt sich mittels LEGO Mindstorms nicht nur das Thema der Robotik an sich vermitteln, sondern es kann auch als Sprungbrett für schwierigeren Stoff oder als Lernmotivation eingesetzt werden. Die Zielgruppe erstreckt sich von Grundschülern bis hin zu Lehrern, die LEGO Mindstorms selbst im Unterrichten einsetzen wollen.

Als wesentliche Kriterien, die für die Auswahl eine Rolle spielen, werden weiters genannt:

- Verständlichkeit und Benutzerfreundlichkeit (sowie Eignung für Einsteiger, wenn man es mit solchen als Zielgruppe zu tun hat)
- Schnelle Entwicklung
- Skalierbarkeit (von einfachen Programmen bis hin zu komplexen Systemen)
- Universalprogrammierung
- Zweckdienliche Steuerung der physischen Einheiten
- Robustheit
- Wartungsunterstützung
- Kosten
- Verträglichkeit mit Lernkurs und Lehrplanzielen
- Einfachheit von Updates und Kosten dafür
- Langlebigkeit

In Zusammenhang mit der LEGO Mindstorms NXT haben NTX-G und leJOS NXJ weite Verbreitung gefunden. Bei NXT-G handelt es sich um eine graphische Programmierumgebung, die im Standardlieferumfang des NXT-Baukastens enthalten ist. leJOS NXJ ermöglicht die Programmierung auf Basis von Java, ist demnach also textbasiert. Beide werden nun anschließend näher vorgestellt.

4.4.2.1 NTX-G

NTX-G stellt einen typischen Vertreter für eine graphische Programmieroberfläche dar und zielt auf Kinder und Erwachsene ohne Programmiererfahrung ab. Vom Benutzer ist kein eigener Code zu schreiben, sondern das Programm wird großteils mittels entsprechender Mausaktionen erstellt. Sind Texteingaben erforderlich, dann geschieht dies dialogbasiert. In Anlehnung an [@deba] erfolgt nun eine auf das Wesentliche beschränkte Einführung in NXT-G.

Nach dem Start des Programms zeigt sich dem Benutzer das folgende Fenster:

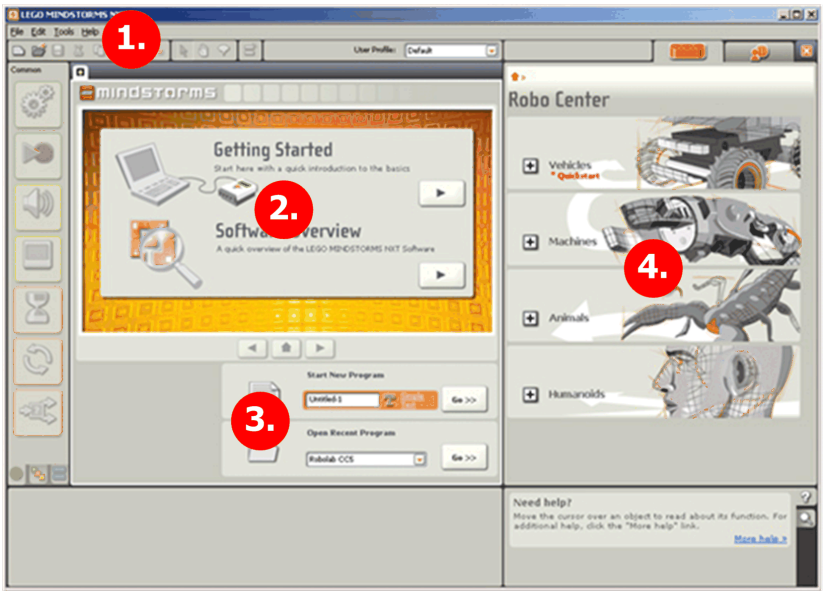


Abbildung 39: Startbildschirm von NXT-G

Hier kann er nun entweder einen Befehl aus dem Hauptmenü bzw. der Symbolleiste ausführen (①), sich eine der Einführungshilfen anschauen (②), ein neues Programm anlegen bzw. ein bestehendes Programm öffnen (③) oder sich eine der Bauanleitungen durchschauen (④). In den meisten Fällen wird sich der Benutzer für das Anlegen eines neuen Programms entscheiden und landet dadurch dann in dem folgenden Fenster, in dem er nun graphisch programmieren kann:

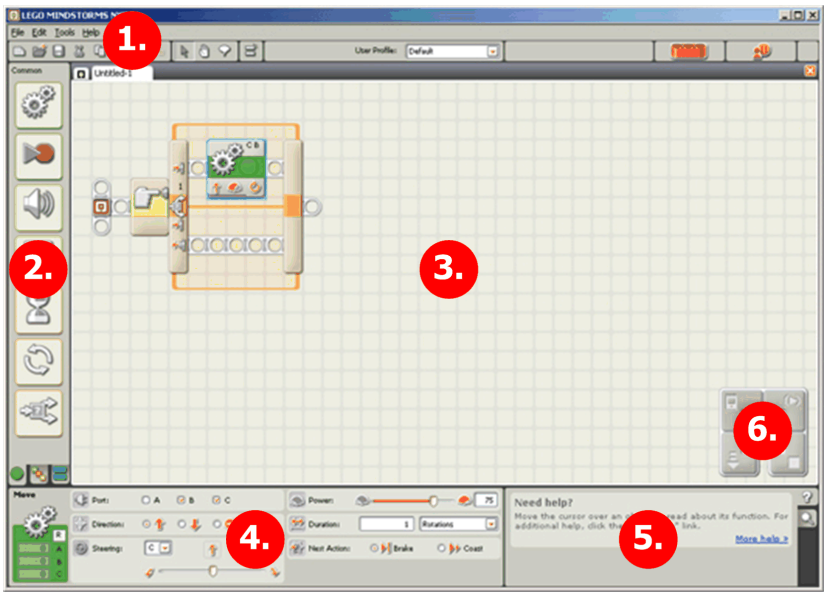


Abbildung 40: Graphische Programmierung in NXT-G

Auch hier kann er wieder einen Befehl aus dem Hauptmenü bzw. der Symbolleiste ausführen (①). Interessanter ist wohl aber, dass der Entwickler nun Blöcke bzw. Kontrollstrukturen aus der linken Symbolleiste (②) per Drag & Drop in das Programmierfeld (③) ziehen kann, deren Eigenschaften in der unteren Dialogbox (④) individuell anpassbar sind. Bei Bedarf kann er sich auch zu den einzelnen Elementen Hilfe anzeigen lassen (⑤). Will der Benutzer sein erstelltes Programm auf den Roboter

übertragen und dort dann ausführen lassen, so stehen ihm dazu auch entsprechende Funktionalitäten zur Verfügung (⑥).

Im Grunde besteht die graphische Programmierung in NXT-G aus vielen kleinen Programmierschritten, die dem folgenden Schema folgen:

Abbildung 41: Ablauf eines Programmierschrittes in NXT-G

Per Drag & Drop wird ein Block bzw. eine Kontrollstruktur in das Programmierfeld gezogen und dort dann entweder mit dem Startblock oder einem bereits vorhandenen Elemente verknüpft. Dies geschieht auf ähnliche Art und Weise wie beim Zusammenbau von LEGO-Modellen in der realen Welt. Zu guter Letzt sind noch die Eigenschaften des Blockes bzw. der Kontrollstruktur gemäß den eigenen Wünschen anzupassen.

Bei den Blöcken – diese führen eine bestimmte Aktion durch – kann der Benutzer zwischen den folgenden wählen:

	Aktion → Wirkung	**Wichtige Eigenschaften**
Move (Bewegung)	Drehung (auch Stoppen) von Servomotor/en → Roboter bewegt sich	• Richtung (vorwärts, rückwärts, Stopp) • Lenkung • Geschwindigkeit • Dauer (endlos, Grade, Umdrehungen, Sekunden)
Record/Play (Aufnahme/Wiedergabe)	Aufnahme laufender bzw. Wiedergabe aufgenommener Aktion/en → Roboter nimmt Aktion/en auf und wiederholt sie bei Bedarf	• Name der Aufnahme bzw. Wiedergabe • Anschlussport/s für Aufnahme
Sound (Geräusche)	Abspielen (auch Stoppen) von Sound bzw. Ton → Roboter gibt Geräusch per Lautsprecher von sich	• Lautstärke • Sounddatei • Tonhöhe und Wiedergabedauer
Display (Anzeige)	Anzeige (auch Säubern) von Bild, Text bzw. Zeichnung → Roboter zeigt Information am LCD an	• Bilddatei • Text • Zeichnungstyp (Punkt, Linie, Kreis)

Tabelle 3: Blöcke in NXT-G

Bei den Kontrollstrukturen – diese wirken steuernd auf den Programmablauf ein – stehen die nachfolgenden zur Auswahl:

	Kontrollaktion	Wichtige Ereignisse bzw. Bedingungen
Wait (Ereignis)	Programm wartet solange, bis bestimmtes Ereignis eintritt	• Zeit: Warte, bis bestimmte Zeitspanne abgelaufen ist • Sensor: Warte, bis gemessener Wert eines bestimmten Sensors im angegebenen Intervall liegt
Loop (Schleife)	Programm wiederholt Aktion/en im Schleifenrumpf solange, bis bestimmte Bedingung zutrifft	• Endlos: Wiederhole endlos • Zeit: Wiederhole, bis bestimmte Zeitspanne abgelaufen ist • Sensor: Wiederhole, bis gemessener Wert eines bestimmten Sensors im angegebenen Intervall liegt • Zähler: Wiederhole, bis Schleifenzähler bestimmten Wert hat
Switch (Verzweigung)	Programm führt je nachdem, ob bestimmte Bedingung erfüllt ist oder nicht, zugehörigen Zweig mit Aktion/en aus	• Sensor: Verzweige je nachdem, ob gemessener Wert eines bestimmten Sensors im angegebenen Intervall liegt oder nicht

Tabelle 4: Kontrollstrukturen in NXT-G

Zu Demonstrationszwecken soll nun das graphische Programm zu folgendem Beispiel in NXT-G wiedergegeben werden: Stellt der Roboter bei der Abfrage des Berührungssensors fest, dass der darauf angebrachte Button gedrückt ist, dann soll er zwei Sekunden lang vorwärts, ansonsten rückwärts fahren. Dieser Vorgang wird genau zehn Mal wiederholt werden.

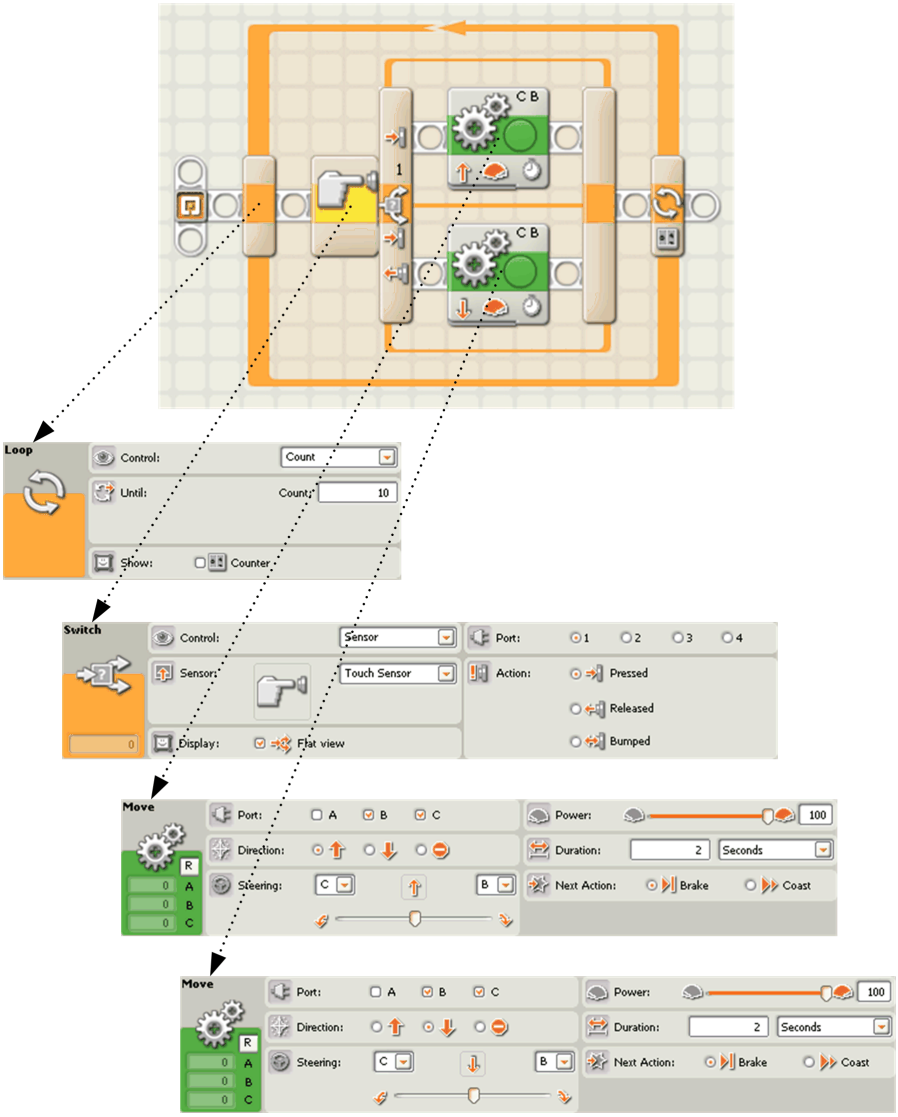


Abbildung 42: Beispielprogramm in NXT-G

Die Verwendung von NXT-G ist zwar einfach und recht intuitiv, jedoch weist es gemäß [Asto07] (S. 139 f.) etliche Mankos auf, wodurch sich dessen Einsatzmöglichkeiten einschränken:

- Die graphische Oberfläche von NTX-G eignet sich nicht besonders für die Erstellung eines größeren Programms. Ab zwanzig Blöcken bzw. Kontrollstrukturen ist dieses nämlich nur mehr schwierig nachzuvollziehen.
- Die Verwendung von benutzerdefinierten Variablen ist in NTX-G zwar grundsätzlich möglich, aber mit Umständen verbunden.
- Diverse nützliche Konzepte bzw. Funktionalitäten, wie z. B. Arrays, Gleitkommazahlen oder trigonometrische Funktionen, fehlen.
- Debugging wird von NXT-G unterstützt, allerdings helfen andere Entwicklungsumgebungen deutlich besser bei der Fehlersuche.
- Für einen Experten ist die Programmerstellung in der graphischen Umgebung wesentlich mühevoller als in einer textbasierten.

4.4.2.2 leJOS NXJ

leJOS NXJ ermöglicht die Programmierung des NXT-Steins mittels der Sprache Java und stellt dafür verschiedene Werkzeuge zur Verfügung. Als die wichtigsten lassen sich gemäß [@lejoa] nennen:

- Firmware: Der Austausch der Firmware durch die mit leJOS NXJ ausgelieferte Version bewirkt, dass im NXT-Stein eine Java Virtual Machine (JVM) installiert wird. Diese ist die Voraussetzung dafür, dass der NXT-Stein entsprechende Java-Programme ausführen kann.
- Bibliothek: Der Zugriff auf die NXT-Bauteile wird über die Klassen der Bibliothek classes.jar ermöglicht. Diese Klassen implementieren in ihrer Gesamtheit das Application Programming Interface (API), das der Entwickler zur Programmerstellung nutzt.
- Linker: Der Linker erzeugt aus dem vom Benutzer erstellten Java-Programm unter Verwendung der Bibliothek eine Binärdatei, die auf den NXT-Stein übertragen und dort dann ausgeführt werden kann.
- Tools: Für den Austausch der Firmware, den Programmupload, das Debugging sowie für diverse andere Funktionen sind im Lieferumfang von leJOS NXJ entsprechend Tools enthalten.

leJOS NXJ installiert beim Firmwareaustausch zwar lediglich eine abgespeckte JVM-Version, diese ist aber immer noch recht leistungsfähig. Nach [Bagn07] (S. 51 ff.) lässt sich jene wie folgt charakterisieren:

- Speicher: Die JVM selbst benötigt ungefähr 27 KB des Speichers, wodurch für Benutzerprogramm und -daten rund 229 KB übrigbleiben. Im Vergleich zur RCX-Version hat sich der verfügbare Speicher somit um ungefähr den Faktor 15 erhöht.
- Geschwindigkeit: In Zusammenhang mit der Robotik hat die Geschwindigkeit keine überaus große Bedeutung. Da selbst die RCX-Version bereits ausreichend schnell war, und die NXT-Version sogar um ungefähr den Faktor 6 schneller sein soll, muss man sich diesbezüglich keine Sorgen machen.
- Gleitkommazahlen: Die JVM erlaubt die Verwendung von Gleitkommazahlen, die wiederum die Verwendung der trigonometrischen Funktionen ermöglichen.
- Threads: Das von der JVM implementierte Thread-Modell ist beinahe vollständig und erlaubt die Synchronisation und Unterbrechung von Threads. Ein einfacher Scheduler ist für die Verwaltung und Steuerung der Threads verantwortlich. Theoretisch können bis zu 255 Threads verwendet werden, wobei zu beachten ist, dass natürlich jeder Speicherplatz belegt.
- Arrays: Die JVM erlaubt nicht nur einfache, sondern auch mehrdimensionale Arrays. Übertreiben sollte man es bzgl. der Dimensionen aber nicht, denn sonst besteht die Gefahr des Speicherüberlaufes.
- Event-Modell: Es wird das in Java übliche Modell für Ereignisse verwendet. Objekte (Event-Listener) registrieren sich dabei bei einem bestimmten Objekt (Event-Source), um von diesem über die auftretenden Ereignisse informiert zu werden und dann darauf entsprechend reagieren zu können.

- Exceptions: Die JVM unterstützt natürlich auch entsprechendes Exception-Handling, wodurch eine saubere Trennung zwischen dem fehlerprüfenden Code und der eigentlichen Programmlogik ermöglicht wird.
- Rekursionen: Der rekursive Methodenaufruf wird von der JVM erlaubt, wenngleich dieser auf maximal zehn Ebenen beschränkt ist. Falls viele lokale Variablen in einer solchen Methode verwendet werden, reduziert sich die Ebenentiefe weiters.
- Garbage Collection: Das Konzept der automatischen Speicherfreigabe von nicht mehr referenzierten Objekten wird von der leJOS NXJ-JVM nicht unterstützt. Somit sollte man mit der Erzeugung neuer Objekte sorgsam umgehen und stets versuchen, nicht mehr gebrauchte Objekte wiederzuverwenden.

Um nun tatsächlich ein Java-Programm schreiben zu können, das einen NXT-Roboter steuert, bedarf es einer entsprechenden Programmierumgebung. Im Grunde ist durch den modularen Aufbau von leJOS NXJ jede Java-Entwicklungsumgebung verwendbar, besonders zu empfehlen ist Eclipse. Dieses ist nicht nur frei verfügbar, sondern es existiert auch ein leJOS NXJ-Eclipse-Plugin, das die Programmierung deutlich angenehmer gestaltet.

Auf die Installation von leJOS NXJ soll nicht näher eingegangen werden, da dieses Thema unter [@lejob], [Schm09] (S. 11 ff.) und [Bagn07] (S. 24 ff.) ausführlich abgehandelt wird. Vielmehr wird in Folge der Aufbau der leJOS NXJ-API in Anlehnung an [Bagn07] (S. 90 ff.) behandelt. Um den Rahmen der Arbeit nicht zu sprengen, ist die Einführung auf das Wesentliche beschränkt.

Die entsprechende Bibliothek unterteilt sich in mehrere Packages, wobei sich als wichtigste nennen lassen:

	Beschreibung
lejos.navigation	Hierin finden sich Klassen, die den Roboter bei der Navigation unterstützen.
lejos.nxt	Hierin finden sich Klassen, die den Zugriff auf den programmierbaren NXT-Stein, die Servomotoren und die Sensoren ermöglichen. Die Klassen kapseln die Bauteile und ermöglichen mittels entsprechender Methodenaufrufe das Ausführen von Aktionen bzw. die Abfrage von Messwerten.
lejos.nxt.addon	Hierin finden sich Klassen, die den Zugriff und die Steuerung von Bauteilen ermöglichen, die nicht standardmäßig dem NXT-Baukasten angehören.
lejos.nxt.comm	Hierin finden sich Klassen, die zur Kommunikation mit dem NXT-Stein vonnöten sind. Als Übertragungstechnik kann hierbei wiederum zwischen USB und Bluetooth gewählt werden.
lejos.nxt.debug	Hierin finden sich Klassen, die beim Debugging unterstützen.
lejos.util	Hierin finden sich Klassen, die oftmals benötigte Funktionalitäten bereitstellen.

Tabelle 5: Wichtigste Packages der leJOS NXJ-API

Von besonderer Bedeutung sind die Zugriffsklassen des Packages lejos.nxt, mittels denen die einzelnen NXT-Bauteile kontrolliert werden können. Die bedeutendsten Klassen werden anschließend überblicksartig vorgestellt und das Verwendungsprinzip anhand des Servomotors und des Berührungssensors genauer beschrieben.

	Beschreibung
Button	Diese Klasse macht den Zugriff auf die Buttons am NXT-Stein möglich. Weiters lassen sich darüber Event-Listener für die Buttons registrieren.
LightSensor	Diese Klasse ermöglicht den Zugriff auf den Lichtsensor. Es kann nicht nur der aktuelle Messwert abgefragt werden, sondern darüber auch die Leuchtdiode des Sensors aktiviert bzw. deaktiviert werden.
LCD	Diese Klasse bietet grundlegende Methoden für die Ausgabe von Texten auf dem LCD des NXT-Steins an.
Motor	Diese Klasse repräsentiert einen Servomotor und ermöglicht dessen Steuerung bzw. Abfrage.
Sound	Diese Klasse ist für das Abspielen von Sounds bzw. Tönen über den Lautsprecher des NXT-Steins verantwortlich.
SoundSensor	Diese Klasse erlaubt den Zugriff auf den Geräuschsensor. Gewählt werden kann dabei zwischen den beiden oben genannten Operationsmodi.
TouchSensor	Diese Klasse repräsentiert den Berührungssensor und ermöglicht dessen Abfrage.
UltrasonicSensor	Diese Klasse wird zum Zugriff auf den Ultraschallsensor verwendet. Über diesen lassen sich Distanzmessungen durchführen.

Tabelle 6: Wichtigste Klassen des Packages lejos.nxt

Die Klasse lejos.nxt.Motor repräsentiert einen Servomotor und enthält für jeden Port (von A bis C) eine statische Instanz. Man erspart sich dadurch die Instanzierung mittels new und kann recht bequem auf die Motoren zugreifen. Auf einem solchen Objekt können dann – wie bei der Objektorientierung üblich – die gewünschten Methoden zur Steuerung bzw. Abfrage des Motors aufgerufen werden. Um einen an Port A hängenden Servomotor in Vorwärtsrichtung zu drehen, sind folgende Versionen denkbar:

```
// Variante 1: Zugriff über neue Instanz
Motor meinMotor = new Motor(MotorPort.A);
meinMotor.forward();

// Variante 2: Zugriff (indirekt) über statische Instanz
Motor meinMotor = Motor.A;
meinMotor.forward();

// Variante 3: Zugriff (direkt) über statische Instanz
Motor.A.forward();
```

Die Klasse lejos.nxt.TouchSensor repräsentiert den Berührungssensor. Bei der Instanzierung eines Objektes dieser Klasse ist anzugeben, an welchem Port (von 1 bis 4) des NXT-Steins der Sensor hängt. Danach kann dann mittels der Methode isPressed abgefragt werden, ob jener gedrückt ist. Quelltextmäßig sieht dies für einen an Port 1 hängenden Berührungssensor wie folgt aus:

```
TouchSensor meinTouchSensor = new TouchSensor(SensorPort.S1);
if (meinTouchSensor.isPressed()) {
  // Berührungssensor gedrückt => Aktionen ausführen ...
}
else {
  // Berührungssensor nicht gedrückt => Aktionen ausführen ...
}
```

Das vorhin unter NXT-G eingeführte und in graphischer Form ausgearbeitete Beispiel soll nun unter Verwendung der leJOS NXJ-API in Java umgesetzt werden. Der folgende Quellcode stellt eine mögliche Lösung dafür dar:

```
import lejos.nxt.*;

public class Roboter {
  public static void main(String args[]) {
    try {
      TouchSensor meinTouchSensor = new TouchSensor(SensorPort.S1);

      for (int i = 0; i < 10; i++) {
        if (meinTouchSensor.isPressed()) {
          Motor.B.forward();
          Motor.C.forward();
        }
        else {
          Motor.B.backward();
          Motor.C.backward();
        }

        Thread.sleep(2000);
      }
    }
    catch (Exception e) {}
  }
}
```

Als Resümee lässt sich festhalten, dass leJOS NXJ ein leistungsfähiges und gut strukturiertes Gesamtpaket zur Programmierung von LEGO Mindstorms NXT unter Java darstellt, das noch dazu kostenlos erhältlich ist. Für bereits erfahrene Java-Entwickler ist die gut dokumentierte API sicherlich rasch verständlich, wodurch einer schnellen Programmentwicklung nichts im Wege steht. Nach [Rink01] (S. 8 ff.) eignet es sich aber auch für eine umfassende Einführung in Java und die objektorientierten Konzepte.

4.5 LEGO Mindstorms im Unterricht

In den letzten Jahren hat LEGO Mindstorms an vielen Schulen und Universitäten Einzug gehalten und dort werden die Roboterbaukästen zur Durchführung unterschiedlichster Projekte genutzt.

Gemäß [Schm09] (S. 6) eignet sich LEGO Mindstorms gut für den fächerübergreifenden Unterricht. Als großer Vorteil wird dabei angesehen, dass es sowohl als Spielzeug als auch zur Simulation realer Vorgänge eingesetzt werden kann. Das Werk liefert viele Vorschläge, wie sich LEGO Mindstorms konkret in unterschiedlichen Unterrichtsfächern einsetzen lässt. Auch [Altm05] (S. 56 ff.) stellt verschiedene Unternehmungen aus dem Ausbildungsbereich vor.

Es werden nun etliche ausgewählte Projekte näher vorgestellt. Sie stehen stellvertretend für die vielen Bemühungen, mit LEGO Mindstorms den Unterricht interessanter und motivierender zu gestalten.

4.5.1 Forschendes Lernen

Der in [Stol07] beschriebene Projektunterricht stellt wohl ein typisches Beispiel für die Verwendung von LEGO Mindstorms zum Erlernen des Programmierens im Rahmen des entdeckenden Lernens dar.

Dieses Projekt wurde an der Polytechnischen Schule Tamsweg in Österreich im Fachbereich Informationstechnologie in der neunten Schulstufe abgehalten. Die Unterrichtseinheiten verteilten sich dabei auf vier Nachmittage zu je drei Schulstunden. Insgesamt nahmen neun Schüler daran teil, die noch nie programmiert hatten und in drei Dreiergruppen zusammengelost wurden.

Nach einer kurzen Einführung in die Handhabung der graphischen Programmierumgebung durch den Lehrer wurden den Schülern zehn Aufgaben (von einfach bis schwer) vorgelegt, die sie eigenständig in ihren Gruppen mit LEGO Mindstorms auszuarbeiten hatten. Ihnen sollte dadurch nicht nur der Einstieg in das Programmieren vermittelt werden, sondern sie sollten auch das selbstständige Lösen praktischer Aufgabenstellungen üben, die Form Gruppenarbeit kennenlernen und das eigenständige Arbeiten erproben.

Zwecks Evaluierung mussten die Unterrichtenden vor und nach dem Projekt einen Fragebogen ausfüllen. Die Auswertung der Antworten zeigte, dass nicht nur der Lehrende, sondern auch seine Schüler sehr positiv resümierten.

Der Lehrer stellte fest, dass sich in jeder Gruppe Spezialisten für die unterschiedlichen Bereiche (Programmierung, Strukturierung der Aufgabenstellung, Roboterbau) entwickelten, bei den Schülern Stolz und Zufriedenheit nach dem Lösen einer Aufgabe feststellbar waren und sich ein positiver Konkurrenzkampf zwischen den Gruppen entwickelte.

Die Unterrichteten schätzten zwar den fachlichen Nutzen für ihre Zukunft als eher gering ein, hatten aber großen Spaß an der Art und Weise des forschenden Lernens und gaben an, im sozialen Bereich durch das Projekt viel dazugelernt zu haben. Sie wünschten sich, dass es zukünftig mehr Projektunterricht nach diesem Schema geben würde.

4.5.2 Jago

Auch das amerikanische Militär setzt bei ihrer Ausbildung auf LEGO Mindstorms, wie das unter [Flow02] beschriebene Projekt zeigt.

Der Roboterbaukasten wird nämlich gemeinsam mit Java in der Militärakademie in West Point eingesetzt, um den Studenten das Problemlösen mit dem Computer, grundlegende Programmierkenntnisse sowie die Konzepte von autonomen Fahrzeugen, eingebetteten Systemen, Sensoren und Computersimulationen zu vermitteln.

Es wurde dazu eigens eine Programmierumgebung namens Jago entwickelt, die es den Auszubildenden ermöglicht, ihre in Java geschriebenen Programme vor dem tatsächlichen Upload in den Roboter in einem graphischen Simulator zu testen. Dank Jago kommen nun mehr als fünfhundert Studenten mit lediglich zwanzig bis dreißig

Roboterbaukästen aus. Des Weiteren ersparen sich diese während der Entwicklungs- und Testphase viel Zeit und Aufwand, da sie ihre Programme bequem im Simulator testen können.

Die kurzfristigen Auswirkungen wurden seitens der Projektbetreiber als sehr positiv beurteilt und es ist davon auszugehen, dass auch die langfristigen beachtlich sein werden. Deshalb werden Jago und dadurch natürlich auch LEGO Mindstorms weiterhin in der Militärakademie in West Point eine wichtige Rolle spielen.

4.5.3 Roberta

LEGO Mindstorms wird aber auch eingesetzt, um Mädchen und Frauen für die Technik zu begeistern. Eines dieser Projekte ist Roberta, das in Anlehnung an [Altm05] (S. 56 ff.) vorgestellt werden soll.

Abbildung 43: Roberta – Mädchen erobern Roboter

Ab 2002 wurden in Deutschland diverse Lernkurse im Rahmen dieses Projekt abgehalten. Als Ziele setzte man sich seitens der Organisatoren, das Interesse von Mädchen an der Technik, der Informatik und der Naturwissenschaft zu wecken und deren Selbstbewusstsein im Umgang mit der Technik zu stärken.

Jeder dieser Kurse begann mit einem einführenden Vortrag in Sachen Robotik. Danach konnten sich dann die Teilnehmerinnen mit den Roboterbaukästen vertraut machen und die ersten Schritte damit unternehmen. Anschließend wurden ihnen unterschiedliche, speziell für Mädchen gestaltete Aufgaben (von einfach bis schwer) vorgelegt, die sie lösen sollten. Zur Steigerung der Motivation durften sie bei der Ausarbeitung der Aufgaben miteinander kommunizieren und zusammenarbeiten. Abgeschlossen wurden die Arbeiten jeweils mit einer Präsentation des gebauten Roboters und einem Bericht dazu.

Als sehr positiv stellte sich dabei heraus, dass es keine externe Bewertung gab, sondern das Ganze nach dem Prinzip der Selbstevaluierung ablief. Zeigte der Roboter das gewünschte Verhalten, dann war die Aufgabe erfolgreich gelöst worden, ansonsten musste noch nachgebessert werden.

Nach den Lernkursen waren von den Teilnehmerinnen entsprechende Feedbackbögen auszufüllen, deren Auswertung ein sehr positives Ergebnis brachte: Die überwältigende Mehrheit hatte Spaß am Kurs (94 %), würde den Kurs auch Freundinnen empfehlen (88 %) und sich einen Folgekurs wünschen (74 %). Berücksichtigt wurde dabei die Meinung von fast 1500 teilnehmenden Personen, darunter 81 % Mädchen.

4.5.4 Wettbewerbe

Nicht vergessen sollte man in Zusammenhang mit LEGO Mindstorms darauf, dass sich auch jede Menge Wettbewerbe etablieren konnten, deren Anwendungsbereiche sich von Roboterfußball bis hin zur Algorithmenoptimierung erstrecken. Diese Wettkämpfe motivieren nicht nur Schüler, sondern auch Erwachsene zur intensiven Auseinandersetzung mit den Baukästen.

Stellvertretend für diese Bewerbe sollen nun die unter [@hand] ausgeschriebenen Wettkämpfe vorgestellt werden. Es handelt sich dabei um die FIRST LEGO League (FLL) und die World Robot Olympiad (WRO).

Die FLL verfolgt die Ziele, Kindern und Jugendlichen zwischen zehn und sechzehn Jahren die Wissenschaft und Technik in sportlicher Atmosphäre näherzubringen, ihnen den Gedanken des Teamgeists zu vermitteln und sie zur kreativen Lösungsfindung zu motivieren. Der Wettbewerb steht dabei jedes Jahr unter einem anderen zeitgemäßen Motto. Die Teams haben nach der Veröffentlichung der Aufgabenstellung acht Wochen Zeit, eine entsprechende Lösung zu erarbeiten. Das Ergebnis dieser Anstrengungen ist dann ein auf Basis von LEGO Mindstorms konstruierter und programmierter Roboter.

Die WRO stellt die Weiterführung des FLL-Bewerbs dar und zielt auf Jugendliche zwischen sechzehn und achtzehn Jahren ab. Der mittels LEGO Mindstorms zu realisierende Roboter ist hier von den Teamgruppen allerdings erst am Tag des Wettkampfes zusammenzubauen, wofür genau zweieinhalb Stunden zur Verfügung stehen. Den Teams ist zwar vorab der Spielfeldaufbau bekannt ist, jedoch werden direkt vor dem Wettbewerb noch zusätzliche Regeln bekanntgegeben, die dann beim Roboterbau zu berücksichtigen sind.

5 Allgemeine Themen zur Vermittlung objektorientierter Konzepte

Dieses Kapitel beschäftigt sich mit Themen, die in Verbindung mit der Vermittlung objektorientierter Konzepte stehen. Es wird darüber diskutiert, ob man im Unterricht von Anfang an auf das objektorientierte Paradigma setzen soll und welche besonderen Probleme und Herausforderungen dabei auftreten. Außerdem werden Vorgehensweisen für den Einsteigerunterricht aufgezeigt und diesbezüglich Tipps dafür gegeben. Zum Abschluss werden unterschiedliche Aspekte von Anfängersystemen betrachtet.

5.1 Objektorientierung zuerst?

War es früher üblich, im Programmierunterricht mit einer Einführung in die prozedurale Programmierung zu beginnen, so scheiden sich heute die Geister, welches Programmierparadigma für den Einstieg am geeignetsten ist. Es sollen deshalb nun im Anschluss die unterschiedlichen Meinungen gegenübergestellt werden.

Das Konzept, die Schüler von Anfang an mit den objektorientierten Prinzipien zu konfrontieren, wird als "objects first" bezeichnet. Gemäß [Lewi00] ist diese Phrase allerdings nicht eindeutig definiert, was Konfliktpotential in sich birgt. So ist auf alle Fälle zwischen der Implementierung von Klassen, die das Verhalten von Objekten definieren, und der Verwendung von Objekten, deren Verhalten durch bestehende Klassen definiert wird, zu unterscheiden. Manche Lehrende bevorzugen den ersten Ansatz für den Einstieg, da gemäß ihrer Meinung die Schüler nur auf diese Art und Weise die objektorientierten Konzepte erlernen können. Die Mehrzahl beginnt allerdings mit der Verwendung von Objekten auf Basis bestehender Klassen.

Weiters betont [Lewi00], dass sich die Prinzipien der prozeduralen und der objektorientierten Programmierung nicht gegenseitig ausschließen. Vielmehr erweitert die Objektorientierung einige Ideen, die sich bereits im prozedularen Paradigma fanden. Als Beispiel sei dazu die modulare Aufteilung der Programmfunktionalitäten in Form von Prozeduren genannt, die beim objektorientierten Ansatz durch die Aufteilung des Codes auf Klassen noch verstärkt wird.

5.1.1 Pro und Kontras

Gemäß [Crut95] sollte das objektorientierte Denken von Anfang an und zyklisch durch alle Klassenstufen gelehrt werden. Als Vorgehensweise dazu wird vorgeschlagen, informell mit realitätsnahen Beispielen zu beginnen und diese dann schrittweise zu formalisieren. Durch die Zunahme der Zahl und der Größe der Objekte lässt sich hierbei nach Belieben die Komplexität der Beispiele steigern.

Als Vorteil seitens der Informatik wird genannt, dass man anfangs mit wenig Begrifflichkeiten auskommt. Im Grunde wird jedes reale Objekt, das Informationen aufnimmt, speichert, verarbeitet und abgibt, einfach mittels eines Informationsobjektes modelliert und die Interaktion zwischen den realen Objekten wird durch entsprechende Kommunikation zwischen den Informationsobjekten aufgelöst. Die Schwerpunkte verlagern sich beim objektorientierten Denken von der Programmierung zur Modellierung.

Des Weiteren wird vorgeschlagen, das objektorientierte Denken möglichst früh im Unterricht zu vermitteln, da Analyse- und Entwurfskompetenzen vorrangig durch praktische Übung erlernt werden. Gutes Modellieren ist demnach Erfahrungssache – beginnt man zu spät damit, dann fehlt den Schülern die Erfahrung. Im Laufe der Zeit lernen die Unterrichteten außerdem noch, dass sich jedes reale Problem mittels unterschiedlicher, gleichwertiger Modelle lösen lässt und es immer mehr als nur eine richtige Lösung gibt.

Auch [Spol95] plädiert dafür, von Anfang an auf die Objektorientierung zu setzen, da es wenig Sinn macht, den Schülern mühevoll die prozedurale Entwicklung beizubringen, um diesen dann bei größeren Projekten jenes Paradigma wieder mühevoll auszutreiben.

Nach [Köll99a] ist genau dieser Paradigmenwechsel auch der Grund dafür, weshalb immer mehr Ausbildungsstätten mit der Objektorientierung beginnen und rät deshalb ebenfalls dazu. Für Schüler scheint es deutlich schwieriger zu sein, sich die objektorientierte Denkweise und einen zugehörigen Programmierstil anzueignen, wenn sie vorher prozedural entwickelt haben. Man nimmt an, dass ein durchschnittlicher Programmierer zwischen sechs und achtzehn Monaten benötigt, um den Paradigmenwechsel zu vollziehen. Andererseits hat die Erfahrung gezeigt, dass die Lernenden bzgl. Verständnis objektorientierter Konzepte wenig Schwierigkeiten haben, wenn sie mit diesen zuerst konfrontiert werden. Daraus lässt sich folgern, dass nicht die Objektorientierung an sich, sondern der oft zu vollziehende Paradigmenwechsel den Schülern Probleme bereitet. Soll ihnen auch die prozedurale Programmierung nähergebracht werden, dann wird geraten, diese erst nach der Objektorientierung durchzunehmen.

[@csis] sieht es ebenfalls als Fehler an, mit der prozeduralen Entwicklung zu beginnen, wenn das Ziel die Vermittlung der Objektorientierung darstellt und argumentiert hierbei ähnlich wie [Köll99a]. Da erfahrene prozedurale Programmierer zwölf bis achtzehn Monate für die Änderung ihrer Sichtweise benötigen, versuchen diese, während der Umstellungsphase die Programmierprobleme weiterhin durch Aufteilung auf Prozeduren anstatt durch Auffinden von Objekten zu lösen. Speziell in Stresssituationen entstehen auf diese Art schlecht konstruierte, implementierte und wartbare Programme.

Gemäß [Schw95] sprechen drei Gründe für die Einführung der Schüler in die Implementierung mittels des objektorientierten Paradigmas, die wie folgt lauten:

- Der Ansatz erlaubt einen zeitgemäßen Informatikunterricht, in dem mächtige Konzepte wie Erweiterbarkeit, Anpassbarkeit, Rekonfiguration, Vererbbarkeit, Kapselung und evolutionäre Softwareentwicklung, aber auch die Anwendungsorientierung anstatt der Vermittlung der technischen Funktionsweise in den Vordergrund rücken.
- Der Ansatz lässt sich bzgl. des didaktischen Prinzips der Fortsetzbarkeit beliebig erweitern und ausbauen.
- Der Ansatz harmoniert vom Stil her mit den grundlegenden, kognitiven Prozessen, die beim Denken, Erkennen und Problemlösen im Gehirn eines Menschen ablaufen.

Laut [Spol98] hat sich die Frage, ob die Objektorientierung im Unterricht gelehrt werden soll, durch ihren Siegeszug von selbst geklärt und muss nun lediglich über die

Frage, wie man jene am besten einführt, diskutiert werden. Auf alle Fälle sind dafür passende Werkzeuge, Sprachen und Beispiele zu wählen, denn nur so kommen die Vorzüge der objektorientierten Programmierung für die Schüler auch wirklich zum Vorschein.

Dieser Auswahl an positiven Meinungen stehen aber natürlich auch kritische gegenüber, die nun in Folge abgehandelt werden.

[Deck94] nennt zehn Gründe, weshalb sich die objektorientierte Programmierung nicht für den Anfängerunterricht eignet. Als wesentliche Argumente gegen den Einstieg mittels Objektorientierung lassen sich folgende nennen: Nach wie vor sind Kenntnisse über Algorithmen vonnöten, die natürlich nicht per se Teil des objektorientierten Konzepts sind, weiters ist der Lehrplan stoffmäßig sowieso schon überfüllt und nur wenig kann daraus gestrichen werden, ohne Einbußen zu haben. Nun noch zusätzlich dieses Paradigma zu unterrichten, würde sowohl die Schüler als auch die Lehrer überfordern.

Gemäß [Heub04] ist der Nutzen der Objektorientierung zur Entwicklung großer komplexer Systeme nicht in Frage zu stellen. Sehr wohl ist allerdings die Sinnhaftigkeit des objektorientierten Einsatzes im Unterricht zu hinterfragen, da man dort eher mit kleinen Problemen zu tun hat, die man mittels eines prozeduralen Ansatzes meistens einfacher lösen kann. Verantwortlich dafür ist vorrangig das aufwendige Begriffssystem der Objektorientierung.

Die Vermittlung der objektorientierten Sichtweise auf Daten eignet sich laut [Hubw05] für den Anfangsunterricht, nicht aber die objektorientierte Programmierung. Hierzu müssten nämlich die Schüler gleichzeitig mit einer Vielzahl von Konzepten, wie z. B. Variablen, Funktionen, Records oder Zeigern, konfrontiert werden, um ihr erstes Programm schreiben zu können, was sie überfordert.

Diese Überforderung sieht auch [Noll07] (S. 8) und schlägt deshalb vor, die grundlegenden Programmierprinzipien mittels des prozeduralen Schemas einzuführen und sich erst danach dem objektorientierten Paradigma zuzuwenden.

Nach [Börs07] liegt der Vorteil des prozeduralen Ansatzes darin, dass sich die Konzepte schrittweise einführen lassen. Durch die enge Verbundenheit der Basiskonzepte ist dies aber bei der Objektorientierung nicht möglich. Schwieriger zu erlernen ist dieses Paradigma zwar nicht, allerdings schwieriger zu lehren.

Auch [Meye04] ist der Meinung, dass die objektorientierte Softwareentwicklung Anfängern gegenüber nur recht mühevoll vermittelt werden kann. Neben den objektorientierten Konzepten müssen nämlich noch zusätzlich die Fähigkeit zur Abstraktion, die Syntax und Semantik der Programmiersprache und die Funktionen der Programmierumgebung erlernt werden.

[Whit07] nennt als Vorteil der prozeduralen Programmierung, dass sich das Problem leichter von oben nach unten in kleinere Schritte zerlegen lässt – man bezeichnet diese Vorgehensweise als top-down –, wodurch sich die Schüler mit der Lösungsausarbeitung leichter tun. Das objektorientierte Paradigma ist zwar enger mit den Vorgängen der realen Welt verbunden, allerdings aufgrund des nicht sequentiellen Ablaufes bei der

Programmausführung schwieriger nachvollziehbar. Aus diesen Gründen produzieren Anfänger mit prozeduralen Sprachen bessere Lösungen als mit objektorientierten.

Unter [Brin04] (S. 37 f.) finden sich bei Interesse weitere kritische Ansichten hinsichtlich des Beginns der Programmierung mittels objektorientiertem Paradigma.

5.1.2 Modellierung vs. Programmierung

In Zusammenhang mit dem Informatikunterricht findet eine Diskussion darüber statt, ob die Entwicklung lauffähiger Programme überhaupt zur Allgemeinbildung beiträgt. [Hubw07] (S. 87 ff.) fasst hier die unterschiedlichen, teils recht kontroversen Meinungen zusammen.

Unumstritten ist nach [Hubw07] (S. 85 ff.) allerdings, dass der Bereich der Modellierung von großer Bedeutung ist und deshalb Teil jedes Informatikunterrichts sein sollte. Aufgrund des Fehlens geeigneter Modellierungstechniken war es bis vor kurzem schwierig, den Schülern das Modellieren auf systematische und angemessene Art und Weise näherzubringen und dadurch missrieten die diesbezüglichen Versuche oftmals zu wenig schülergerechten, rein philosophischen Exkursen. Mittlerweile steht aber dank UML eine Modellierungssprache zur Verfügung, die eine altersgerechte Modellierung einfacher Sachverhalte ermöglicht und sich dadurch für den Schulunterricht eignet.

Speziell bei der Objektorientierung kommt es zu einer starken Vermischung zwischen Modellierung und Programmierung. In Bezug auf die Allgemeinbildung ist besonders die objektorientierte Modellierung bedeutend für den Unterricht. Die Implementierung der auf diese Weise gewonnenen Modelle liefert jedoch gemäß dem oben genannten Werk kaum einen Beitrag zur Allgemeinbildung.

[Schu03] (S. 28) sieht die Modellierung ebenfalls als Schwerpunkt der Objektorientierung. Besondere Bedeutung kommt der objektorientierten Modellierung deshalb zu, weil die Abbildung der realen Welt mittels entsprechender, objektorientierter Verfahren der menschlichen Denkweise sehr nahe kommt, was ein Alleinstellungsmerkmal der Objektorientierung darstellt. In Bezug auf die Implementierung gibt es allerdings wenig Unterschied im Vergleich mit den anderen Programmierparadigmen. Auch dies spricht dafür, die Modellierung und nicht die Programmierung im Unterricht in den Vordergrund zu stellen.

[Diet07] plädiert für den Ansatz "models first", nämlich, dass die objektorientierte Modellierung von Beginn an im Informatikunterricht gelehrt wird, wobei die Schüler nicht nur mit der Modellierungssprache, sondern auch gleich mit der entsprechenden Modellierungstechnik konfrontiert werden sollten. Dadurch soll bewirkt werden, dass die Modellierung von den Lernenden als zentrales Werkzeug zum Problemlösen eingesetzt wird.

Des Weiteren sollte die Modellierung dem Ansatz "strictly objects first" folgend bei den Objekten beginnen und sich mit deren Eigenschaften, Methoden und Beziehungen beschäftigen. Der Unterschied zwischen den beiden Begrifflichkeiten Klasse und Objekt ist dabei anfangs zu vernachlässigen. Der Klassenbegriff sollte erst später eingeführt werden.

Im Unterricht ist außerdem darauf zu achten, dass die Systematik in Zusammenhang mit der Modellierung für die Schüler nachvollziehbar ist. Weiters sollten ihnen stets Alternativen aufgezeigt werden, damit sie die Modellierung als kreativen Prozess kennenlernen, der unterschiedliche Lösungen zulässt. Um ein realistisches Bild vom Problemlösen in der Informatik zu vermitteln, sollten die Schüler auch mit Anforderungsänderungen konfrontiert werden, die eine nachträgliche Anpassung ihrer entworfenen Modelle nach sich ziehen.

Zusätzlich ist zu beachten, dass für die Schüler ein Zweck hinter der Modellierung erkennbar wird. Das Modell muss demnach angewendet werden, was im Regelfall durch die Ausführung eines auf Basis des Modells entwickelten Programms erreicht wird. Die Implementierung ist aber natürlich speziell für Einsteiger eine nicht gerade triviale Aufgabe.

Auch nach [Schu04] (S. 195 ff.) kann eine Einführung in die Informatik mit der objektorientierten Modellierung beginnen, wenngleich jene für Anfänger keinesfalls einfach verständlich ist. Erschwert wird das Ganze noch dadurch, dass zentrale Begriffe zum Einsatz gelangen, die für Schüler von Haus aus mit einer anderen Bedeutung belegt sind. Für den Entwurf werden Klassendiagramme und Struktogramme vorgeschlagen. Als wesentlicher Nachteil wird angeführt, dass viele bedeutsame Konzepte der Informatik fast zeitgleich eingeführt werden müssen, wodurch die anfängliche Vermittlung der objektorientierten Modellierung ein schwieriges Unterfangen darstellt. Nach [Schu04] (S. 216 ff.) eignet sich jene aber hervorragend zur fachlichen Vertiefung. Als Beispiel werden dazu Polymorphie und Entwurfsmuster genannt.

Zusammenfassend lässt sich festhalten, dass die Modellierung nicht nur in Zusammenhang mit dem objektorientierten Programmierparadigma, sondern allgemein als Verfahren zur systematischen Problemlösung für den Informatikunterricht von großer Bedeutung ist. Die Vermittlung entsprechender Kompetenzen darf deshalb in der Ausbildung nicht zu kurz kommen und schon gar nicht fehlen.

5.2 Probleme und Herausforderungen

Beim Erlernen des objektorientierten Paradigmas treten auf der Schülerseite zumeist recht ähnliche Probleme auf, bei der Vermittlung dieses Programmieransatzes stellen sich besondere Herausforderungen an die Lehrperson. Auf beide Punkte wird nun im Anschluss eingegangen.

5.2.1 Anfängerprobleme und -fehler

Die Probleme und Fehler von Programmieranfängern lassen sich in zwei Kategorien unterteilten: Und zwar in jene, die allgemein beim Erlernen einer Programmiersprache auftreten, und in solche, die in Bezug auf die objektorientierte Programmierung typisch sind.

Zu ersterer Klasse gehören dabei die Probleme, die mit dem Erlernen der Syntax einer Programmiersprache in Zusammenhang stehen. So ist das Setzen der abschließenden Semikolons in Java, aber auch anderen Sprachen, gemäß [Hadj98] anfangs für die Schüler schwierig zu erfassen und eine häufige Fehlerquelle, weshalb das Kompilieren

der Klassen scheitert. Weiters werden Schwierigkeiten mit Schleifen, Arrays und Bedingungen sowie Probleme bei der Verwendung der Programmierumgebung genannt.

Zusätzlich zu den vorhin genannten Punkten ist es nach [Sart05] für Programmierneulinge auch nicht leicht, sich die große Anzahl an unterschiedlichen Anweisungen einzuprägen und diese auch richtig einzusetzen. Die Formulierung der Lösung einer Aufgabe in Algorithmenform mittels einer Programmiersprache ist demnach speziell für Anfänger eine große geistige Herausforderung. Dazu kommt, dass die meisten Programmierumgebungen für Experten entwickelt wurden und diese dadurch Neulinge bei ihren ersten Schritten nicht sonderlich unterstützen.

Nach [Börs07] können erfahrene Entwickler außerdem auf eine große Auswahl an Programmiermustern zurückgreifen und durch kombinierte Anwendung derselben komplexe Aufgaben lösen. Anfänger hingegen haben deutlich weniger solche Pläne im Kopf, die dann meist noch dazu auf einzelne Sprachen beschränkt sind und haben – bedingt durch ihre mangelnde Erfahrung – mit dem Zusammensetzen solcher Programmiermuster zur Bewältigung komplexerer Aufgaben Probleme. Das führt dann zu Schwierigkeiten beim Lösungsentwurf. Da bei der Objektorientierung die Pläne und Strukturen orthogonal sind, verstärkt sich bei der Verwendung dieses Paradigmas das eben genannte Problem noch.

Bei Anfängerproblemen in direktem Zusammenhang mit den objektorientierten Konzepten ist gemäß [Humb06] (S. 156), [Schu03] (S. 161) und [Schu08] (S. 4, S. 7) für die Schüler der Unterschied zwischen den Begriffen Klasse und Objekt anfangs nur sehr schwer nachvollziehbar. Fälschlicherweise sehen viele Neulinge die Klasse lediglich als Synonym für Objekt oder als Oberbegriff für mehrere Objekte an. Diese Missinterpretation lässt sich dabei teilweise durch das Phänomen der Ähnlichkeitshemmung erklären, da die beiden stark miteinander verwandten Begriffe oftmals fast gleichzeitig im Unterricht vorgestellt werden.

Auch [Holl97] zählt eine Reihe typischer Missverständnisse auf. So interpretieren manche Programmiernovizen ein Objekt irrtümlicherweise als Wrapper um eine Instanzvariable herum. Diese Fehlannahme lässt sich allerdings leicht ausräumen, indem in Einführungsbeispielen stets mehr als eine Instanzvariable verwendet wird. Andere wiederum sehen in einem Objekt ein einfaches Record, also eine Struktur zum Verbund von Daten. Hierbei wird empfohlen, bei den einführenden Lehrbeispielen auch Objekte zu verwenden, die in Abhängigkeit ihres aktuellen Status auf einen Methodenaufruf reagieren. Des Weiteren wird von manchen Anfängern auch angenommen, dass die Arbeit in Methoden ausschließlich durch Zuweisungen erledigt werden kann. Auch dieser falschen Annahme lässt sich mittels Demonstrieren entsprechender Lehrbeispiele recht einfach begegnen. Für die bereits vorhin aufgezählten Probleme bzgl. Unterscheidung zwischen Klasse und Objekt wird vorgeschlagen, in Übungsbeispielen stets mit mehreren Instanzen einer Klasse zu arbeiten. Als letzter wesentlicher Punkt wird die Verwechslung zwischen der Identität eines Objektes und der Zugriffsvariable darauf genannt, was eine Reihe weiterer Missverständnisse zur Folge hat. In diesem Zusammenhang wird empfohlen, die Schüler entsprechende Programmierexperimente durchführen zu lassen, wie z. B. die gleichzeitige Referenzierung des gleichen Objektes über unterschiedliche Zugriffsvariablen.

Weitere Probleme werden in [Gran05] (S. 62 ff.) und [Gran06] aufgezählt. Hier wurden die Schüler mittels LEGO Mindstorms und leJOS in die objektorientierten Konzepte eingeführt. Vielen Schülern schien der Unterschied zwischen der Deklaration und der Instanzierung bzw. Zuweisung eines Objektes nicht klar gewesen zu sein, denn es traten häufig Fehler auf, die darauf zurückzuführen waren. Auch das Unterscheidungsproblem zwischen Klasse und Objekt machte sich auf Seiten der Schüler bemerkbar. Die Verwendung des Punktoperators in Java sowie die korrekte Parameterübergabe an Methoden bereitete den Lernenden ebenfalls große Schwierigkeiten, wobei das erstgenannte Problem vor allem darauf zurückzuführen ist, dass die Schüler oftmals das Zusammenspiel zwischen den einzelnen Objekten nicht verstanden. Im Allgemeinen bereiteten den Lernenden die Syntax und Semantik von Java größere Schwierigkeiten. So wurden häufig Semikolons vergessen oder falsche Klammertypen verwendet. Das Auskommentieren von Quellcode oder das Ergänzen von eigenen Kommentaren funktionierte ebenfalls nicht problemlos. Teilweise wurde von den Schülern auch der Sinn des Programmuploads auf den LEGO Mindstorms-Roboter nicht verstanden. So transferierten manche das Programm auf den RCX-Stein, änderten anschließend ihr lokales Programm und wunderten sich, weshalb der Roboter noch nicht auf diese Änderungen reagierte. Als problematisch stellte sich auch die eigenständige Verwendung der API-Dokumentation heraus, da den Schülern hierfür die notwendige Erfahrung fehlte.

Nach [Diet05a] analysieren die Programmieranfänger zwar eine zu lösende Aufgabenstellung mit Hilfe von Objektdiagrammen recht problemlos, allerdings scheitern sie dann oftmals bei der Umsetzung jener in ein Programm. Als Gründe hierfür werden aufgezählt, dass die Vorstellung der Schüler hinsichtlich der Möglichkeiten und Einschränkungen des Computers mangelhaft ist und ihnen die Zerlegung des Gesamtproblems in einzelne Programmschritte große Schwierigkeiten bereitet.

Gemäß [Bena08] sind die objektorienterten Konzepte der Kapselung, der Vererbung und des Polymorphismus selbst dann für Schüler schwierig zu verstehen, wenn sie bereits Erfahrung mit anderen Paradigmen mitbringen. Als besonders problematisch stellte sich heraus, dass der Programmfluss nicht sequentiell und dadurch wesentlich schwieriger nachvollziehbar ist. Weitere im Werk genannte Probleme werden hier nicht angeführt, da sie weit über das Anfängerniveau hinausgehen.

Abschließend sei noch auf [Pill09] verwiesen. Dieses Werk beschäftigt sich mit von Neulingen häufig gemachten objektorientierten Entwurfsfehlern und liefert auch viele interessante Literaturverweise. Genannt werden darin Syntaxfehler bei der Verwendung von UML, falsch identifizierte Klassen, Datenelemente, Methoden, Assoziationen und Vererbungsverknüpfungen, das Missverstehen des Vererbungs- und des Abstraktionsprinzips sowie das Nichtverwenden des Polymorphismus, obwohl es aus Entwurfssicht sinnvoll wäre.

5.2.2 Herausforderungen für Lehrende

Immer wieder liest man, dass das Erlernen der Objektorientierung im Vergleich zu anderen Programmierparadigmen nicht schwieriger ist, sehr wohl aber die Vermittlung der zugrundeliegenden Prinzipien eine große Herausforderung für den Lehrenden darstellt.

Einige der Schwierigkeiten wurden bereits vorhin genannt und sollen nun einleitend nochmals aus Lehrersicht dargestellt werden. Zu beachten ist dabei, dass die im Folgenden genannten Punkte zusätzliche Herausforderungen für den Unterrichtenden darstellen und die Vermittlung der eigentlichen objektorientierten Konzepte deutlich verkomplizieren.

Laut [Deck94] müssen die Schüler auch in Sachen Algorithmen geschult werden. Gemäß [Hubw05] sind die Lernenden zusätzlich zu den objektorientierten Prinzipien noch mit einer Vielzahl von Konzepten, wie z. B. Variablen, Funktionen, Records oder Zeigern, zu konfrontieren. Nach [Meye04] ist außerdem die Fähigkeit zur Abstraktion zu schulen und weiters sind die Syntax und Semantik der Programmiersprache und die Funktionen der Programmierumgebung zu lehren. Dazu kommt gemäß [Sart05] noch die große Zahl an unterschiedlichen Anweisungen, welche die Schüler für die Implementierung benötigen. Erschwert wird das Ganze laut [Börs07] dadurch, dass – bedingt durch die enge Verbundenheit der objektorientierten Basiskonzepte – diese nur schwer schrittweise eingeführt werden können.

Auch das Fehlen von entsprechenden Unterrichtshilfen erschwert die Einführung in die Objektorientierung. [Brin04] (S. 41 ff.) bemängelt in diesem Zusammenhang die Diskrepanz zwischen den fachdidaktischen Forschungsthemen zur Objektorientierung und den schulpraktischen Erwägungen und fordert entsprechende Schnittstellenkonzepte. Bzgl. der objektorientierten Modellierung werden die folgenden Punkte als fehlend aufgezählt:

- Konzepte für die Vermittlung von Basiskonzepten
- Unterrichts- und Übungsbeispiele
- Unterrichtsgeeignete Informatiksystemunterstützung beim Erlernen der Basiskonzepte
- Empfehlungen zur Strukturierung des Lehr-Lern-Prozesses

Hierbei muss jedoch auch angemerkt werden, dass es gerade in den letzten Jahren Fortschritte bzgl. der Bereitstellung entsprechender Unterrichtshilfen gab, wenngleich man sich noch nicht am Ziel wähnen kann.

Des Weiteren stellt das Finden des passenden Mittelmaßes – einerseits gilt es die grundlegenden Konzepte zu vermitteln, andererseits sollen die Schüler diese auch in einer bestimmten Programmiersprache umsetzen lernen – eine große Herausforderung für den Lehrer dar. Nach [Humb06] (S. 158 f.) sollten in der Ausbildung überhaupt die Zielorientierung und die Planung der Problemlösung im Vordergrund stehen, nicht jedoch aber die Sprachkonstrukte einer Programmiersprache.

Nicht leichter wird die Aufgabe für den Lehrenden dadurch, dass das gleiche Konzept in den einzelnen Programmiersprachen unterschiedlich bezeichnet wird – die folgende Tabelle gemäß [Börs07] gibt Auskunft darüber – und dass dann oftmals auch noch die Syntax von diesen Begriffen abweicht. Es wird deshalb empfohlen, zwischen Konzepten, Notationen und Terminologien im Unterricht klar und sauber zu unterscheiden.

	UML	Java	C++
Attribut	attribute	(field) variable	data member bzw. (instance/reference) variable
Methode	operation	method	member function
Eigenschaft	feature	member	member
Methoden-implementierung	method	method definition (body)	(member) function definition
Subklasse	child	subclass	derived class bzw. child class
Superklasse	parent	superclass	base class
Vererbung	generalization	inheritance	Inheritance

Tabelle 7: Unterschiedliche Begriffe für objektorientierte Konzepte

[Börs07] betont weiters, dass die Wahl sinnvoller und anschaulicher Beispiele von großer Bedeutung ist. Es reicht hierbei nicht aus, die gängigen Lehrbeispiele lediglich in eine objektorientierte Programmiersprache zu transferieren, da dadurch dann oftmals die Vorteile der objektorientierten Konzepte nicht zur Geltung kommen und im schlimmsten Fall deren Verwendung den Schülern sogar wesentlich aufwändiger und komplizierter erscheint. Da den Lehrbeispielen in frühen Lernphasen große Bedeutung zukommt und diese den Lernenden als Vorlagen für die eigene Arbeit dienen, stellt deren sorgfaltige Auswahl eine große Herausforderung für den Unterrichtenden dar. Konkret wird in diesem Zusammenhang z. B. vor einer Einführung in das Klassenkonzept mittels java.lang.Math gewarnt, da diese Java-Klasse als statisch deklariert ist und dadurch nicht die grundlegenden Eigenschaften einer üblichen Klasse aufweist.

Jedoch nicht nur das Auswählen sinnvoller Beispiele, auch die Wahl einer geeigneten Programmiersprache und -umgebung ist gemäß [Köll99a] und [Köll99b] für die Vermittlung der objektorientierten Konzepte von enormer Bedeutung. Da viele der eingesetzten Sprachen für den Anfangsunterricht zu komplex sind und die Entwicklungsumgebungen oftmals einen Anfänger mehr verwirren als diesen bei den ersten Schritten zu unterstützen, muss der Lehrende in Bezug auf die Programmiersprache und -umgebung sehr sorgfältig auswählen.

Die wohl größte Herausforderung ist es für viele Lehrende allerdings, einen Lehrstoff vermitteln zu müssen, zu dem die eigenen Kenntnisse mangelhaft sind. Diese Aussage mag provokant klingen, der entsprechende Sachverhalt lässt sich jedoch zumindest für die deutschen Lehrer anhand der 2005/2006 durchgeführten und unter [Kohl06] veröffentlichen Untersuchung nachvollziehen. Obwohl die Objektorientierung im Informatikunterricht und in den zugehörigen Lehrplänen einen wichtigen Platz eingenommen hat, sind nur wenige Lehrer mit den objektorientierten Basiskonzepten ausreichend vertraut und schätzen ihr eigenes Wissen in diesem Bereich als umfassend ein. Dies ist aber eine notwendige Voraussetzung dafür, dass man den Schülern die Objektorientierung und ihre Prinzipien näherbringen kann. Verbesserungen im Ausbildungs- und Weiterbildungsbereich der Lehrer sind hier also dringend notwendig.

5.3 Vorgehensweisen und Tipps

Es gibt unterschiedliche Ansätze, die im Anschluss beleuchtet werden sollen, um Einsteiger an die objektorientierten Prinzipien heranzuführen. Des Weiteren werden danach Ratschläge für den objektorientierten Anfangsunterricht geliefert.

5.3.1 Vorgehensweisen zum Einstieg in Objektorientierung

[Schu03] (S. 14 ff.) nennt fünf verschiedene Konzepte für den Anfangsunterricht:

- System warten: Bei diesem Ansatz wird den Schülern ein fehlerhaftes Programm mit unvollständiger Dokumentation vorgelegt. Sie müssen anfangs den Programmaufbau analysieren, dann die darin enthaltenen Fehler ausfindig machen und beheben sowie die zugehörige Dokumentation vervollständigen. Das vorrangige Ziel dabei ist es, dass die Lernenden die Programmstruktur, die Funktionen der Programmierumgebung und diverse typische Programmierkonzepte, wie z. B. den Algorithmenbegriff oder Prozeduren, kennenlernen. Im Anschluss an diese Wartungstätigkeiten wird die gemeinsame Entwicklung eines ähnlichen Programms empfohlen.
- Projektorientierter Einstieg: Bei diesem Ansatz implementiert der Lehrer gemeinsam mit den Schülern ein Programm. Im Rahmen der Entwicklung werden dabei Schritt für Schritt die objektorientierten Konzepte eingeführt und die Funktionen der Programmierumgebung bzw. die zur Implementierung notwendigen Sprachkonstrukte vorgestellt.
- Formulardesigner nutzen: Bei diesem Ansatz beginnen die Schüler mit der Erstellung der graphischen Benutzeroberfläche eines Programms. Es muss dazu natürlich eine passende, integrierte Entwicklungsumgebung zum Einsatz gelangen, in der die Lernenden die visuellen Komponenten per Maus den jeweiligen Formularen hinzufügen können. Die Eigenschaften der Komponenten lassen sich für gewöhnlich über Dialogboxen anpassen. Man spricht in diesem Zusammenhang von der graphischen Programmierung. Normalerweise kann nicht das gesamte Programm auf diese Art erstellt werden, sondern es muss auch ein eigener Quellcode geschrieben werden. Der große Vorteil dieser Vorgehensweise ist, dass die Schüler rasch einen sichtbaren Erfolg zu verzeichnen haben.
- Bibliothek nutzen und anschließend erweitern: Bei diesem Ansatz beginnen die Schüler damit, kleinere Programmieraufgaben mit Hilfe einer vorgefertigten Bibliothek zu lösen. Mit jeder Übung steigert sich dabei die Schwierigkeit und Schritt für Schritt werden neue Konzepte eingeführt. Sobald die Lernenden ausreichend mit der Nutzung der Bibliothek vertraut sind, wird deren Erweiterung um neue Funktionalitäten verlangt.
- Sprachkurs: Bei diesem Ansatz erfolgt die Einführung der Schüler in die Objektorientierung mittels eines Sprachkurses. Statt einer prozeduralen Programmiersprache wird hier nun eine objektorientierte gelehrt. Manchmal erfolgt die Einführung in die grundlegenden Programmierkonzepte auch weiterhin mittels einer prozeduralen Sprache und wird erst danach mit einer objektorientierten begonnen. Zu beachten ist, dass solche Kurse zumeist das Ziel, nämlich den Schülern die objektorientierte Denkweise zu vermitteln, verfehlen.

Auch [Schu08] (S. 6 ff.) beschreibt verschiedene Herangehensweisen zum Einstieg von Schülern der elften Schulstufe in die Objektorientierung und unterscheidet dabei zwischen folgenden:

- Klassen- vor Objektbegriff: Bei diesem Ansatz werden die Schüler zuerst mit dem Klassen- und unmittelbar danach mit dem Objektbegriff konfrontiert. Als großer Vorteil dieser Variante lässt sich nennen, dass den Lernenden die beiden zentralen Konzepte der Objektorientierung rasch vorgestellt werden und sie gleich darauf erste objektorientierte Programmierschritte unternehmen können. Nachteilig ist allerdings, dass für die Schüler bei vorheriger Vorstellung des Klassenbegriffes im Rahmen der Modellierung der Bezug zur realen Welt verloren geht. Weiters ist es für sie – bedingt durch die zeitnahe Einführung ähnlicher Begrifflichkeiten – schwierig, den Unterschied zwischen Klasse und Objekt zu verstehen.
- Programmierumgebungen ohne Klassenbegriff: Bei diesem Ansatz gelangt eine Programmierumgebung zum Einsatz, die auf den Klassenbegriff vollständig verzichtet. Den Schülern steht in einem solchen Entwicklungstool zumeist eine entsprechende, graphische Oberfläche zur Verfügung. Über diese können sie dann per Mausklick Objekte aus vordefinierten Klassen zu einer Art Programm zusammenstellen. Jedem Objekt wird dabei ein global eindeutiger Bezeichner zugewiesen, der auch für den Nachrichtenversand zwischen den Objekten verwendet wird. Der Vorteil dieses Ansatzes ist, dass die Schüler recht schnell zu sichtbaren Ergebnissen gelangen, was sich motivierend auf deren Arbeit auswirkt. Als großer Nachteil lässt sich allerdings anführen, dass zumeist keine Objekte aus der realen Welt modelliert und verwendet werden können, wodurch die Lernenden dann möglicherweise einen falschen Eindruck vom Objektbegriff bekommen.
- Strictly Objects and Models First: Bei diesem Ansatz wird im Unterricht von Anfang an eine objektorientierte Programmiersprache verwendet. Den Schülern wird hier zuerst der Objekt- und erst später dann der Klassenbegriff vorgestellt. Der anfängliche Schwerpunkt liegt bei dieser Variante auf dem objektorientierten Modellieren, also dem Erkennen realer Objekte und deren Repräsentation mittels entsprechender, virtueller Modelle. Durch die zeitversetzte Einführung der beiden Begriffe wird nun zwar das Problem der Ähnlichkeitshemmung umgangen, allerdings können von den Lernenden in der ersten Phase keine herzeigbaren Programme entwickelt werden – es werden lediglich nicht ausführbare Objektdiagramme produziert –, was wenig motivierend auf diese wirkt.

Eine weitere Vorgehensweise, die den Schwerpunkt der Einführung auf die Modellierung verlagert, nennt [Diet07]. Das Unterrichtskonzept läuft dabei in den folgenden fünf Phasen ab:

- Einführung der Objekte: Am Beginn steht die Vorstellung des Objektbegriffes, wobei jener mittels Objektspielen den Schülern nähergebracht wird. Danach erfolgt mit Hilfe von Objektdiagrammen eine Einführung in die Modellierung. Zu beachten ist dabei, dass die Lernenden hierbei ihre virtuellen Modelle ständig mit der Realität abgleichen. Im Anschluss daran kann dann durch Ableitung des Klassendiagramms aus den Objektdiagrammen der Klassenbegriff eingeführt werden. Es folgt eine Festigungsphase, in der ähnliche Beispiele gemeinsam in Gruppen ausgearbeitet werden. Abgeraten wird davon, die Schüler bereits zu diesem Zeitpunkt mit dem Vererbungsprinzip zu konfrontieren.

- Methoden entwerfen: Beim Methodenentwurf geht man ebenfalls wieder von den Objekten aus. Weiters steht auch hier die Modellierung im Vordergrund. Wichtig ist, dass dabei eine Schreibweise angewendet wird, durch welche die Objekte, ihre Beziehungen zueinander und der Kontrollfluss nachvollziehbar dargestellt werden. Zur Visualisierung der Interaktionen zwischen den Objekten eignen sich wiederum Objektspiele, wobei sich durch Schließen oder Verbinden der Augen der daran teilnehmenden Schüler die Eingeschränktheit der Objekte noch verstärkt wiedergeben lässt. Alternativ zu solchen Objektspielen bietet sich Story Driven Modeling an.
- Ablauf von Methoden verstehen: In Anschluss an den Methodenentwurf ist von den Schülern die Funktionsweise eines bereits existierenden Programms nachzuvollziehen. Dabei ist zu überprüfen, ob die Lösung auch mit der aktuellen Aufgabenstellung korreliert. Außerdem sollen auf diese Art und Weise mögliche Fehler aufgedeckt werden. Zur objektorientierten Darstellung des Programmablaufes empfiehlt sich hierbei die Verwendung von Klebezettel und einer Kamera.
- Projekt durchführen: Von den Lernenden ist dann im Rahmen eines kleinen Projektes ein aus mehreren Klassen und Methoden bestehendes Programm in Gruppenarbeit zu erstellen. Idealerweise sollte das entwickelte Endprodukt für die Schüler angreifbar sein. Programmierbare Roboter, im Speziellen LEGO Mindstorms, eignen sich demnach bestens für solche Projekte.
- Textuelle Programmiersprache erlernen: Hat man sich bei der Modellierung für eine graphische Modellrepräsentation entschieden, dann sollte man im letzten Schritt den Schülern noch eine textuelle Programmiersprache näherbringen. Es entsteht hier zwar ein Bruch in der medialen Darstellung, durch die vorangegangen Übungen fällt dieser aber weit weniger stark als bei anderen Lehransätzen aus. Empfohlenerweise sollten hierbei Tools aus dem Bereich des Computer-Aided Software Engineerings (CASE) zum Einsatz gelangen, die den Schülern die automatische Generierung des Quelltextes bzw. von Teilen davon ermöglichen.

Zu guter Letzt sollte festgehalten werden, dass zum Einstieg in die Objektorientierung unterschiedliche Vorgehensweisen denkbar sind. Wie sooft gilt auch hier: Viele Wege führen ans gewünschte Ziel.

5.3.2 Konkrete Tipps für Unterricht

[Beck01] sieht es als besonders wichtig an, dass den Schülern die wichtigsten Konzepte von Anfang an vorgestellt werden. Nur so kann nämlich erreicht werden, dass diese durch intensives Üben gefestigt werden. Werden sie erst zu einem späteren Zeitpunkt eingeführt, dann verkürzt sich diese Übungsphase bzw. bei Zeitknappheit werden die Konzepte dann oftmals überhaupt nur mehr recht oberflächlich behandelt.

Laut [Diet05b] können viele der aufwändigen Begriffe der Objektorientierung im Schulunterricht überhaupt vollständig vernachlässigt werden. Den Schülern sollten lediglich die wesentlichen, objektorientierten Prinzipien nähergebracht werden. Das entsprechende Werk liefert hierzu einen möglichen Ansatz.

In Zusammenhang mit der Vererbung warnen [Noll07] (S. 40) und [Diet07] davor, die Schüler zu früh damit zu konfrontieren, da sie sehr komplex ist und viel Basiswissen

erfordert. Das Vererbungskonzept sollte demnach erst eingeführt werden, nachdem die Lernenden das Klassen- und Objektkonzept verstanden und eigenständig Klassen erstellt haben. Außerdem sollten sie über die Objektinteraktion Bescheid wissen. Bei Nichtbeachtung dieser Punkte drohen sich sonst nur schwer behebbare Fehlvorstellungen in den Köpfen der Schüler zu manifestieren.

Nach [Reic05] (S. 147 f.) ist besonders darauf zu achten, dass die Schüler nicht gleich am Beginn von den komplizierten Sprachen und Entwicklungsumgebungen abgeschreckt werden. Zur Vermittlung der objektorientierten Basiskonzepte muss kein Programmiersystem aus dem professionellen Bereich verwendet werden, ein speziell für Anfänger entwickeltes reicht vollkommen aus. Konkret beschäftigt sich das Werk mit einer Programmierumgebung für Anfänger namens Kara.

[Köll01] liefert verschiedene Richtlinien, die bei der Vermittelung der objektorientierten Programmierung zu beachten sind. Die meisten dieser Punkte sind als unabhängig von BlueJ zu betrachten, wenngleich deren Einhaltung ohne Verwendung dieser Java-Programmierumgebung schwierig zu bewerkstelligen sein dürfte:

- Objekte zuerst: Am Beginn des objektorientierten Programmierunterrichts sollten tatsächlich auch die Objekte stehen. In den traditionellen Entwicklungsumgebungen ist genau das aber meist schwierig zu bewerkstelligen, da zur Erzeugung und Verwendung von Objekten auch jede Menge anderer Sprachkonzepte bekannt sein müssen. Hier spielt BlueJ eine seiner Stärken aus, denn Objekte können in dieser Programmierumgebung interaktiv erstellt und verwendet werden. Somit können die Objekte tatsächlich am Beginn der Einführung stehen.
- Beginne nicht mit leerem Bildschirm: Oftmals wird beim Lehren der objektorientierten Programmierung der Fehler gemacht, dass bei Null begonnen wird und die Schüler dadurch anfangs vor einem leeren Bildschirm sitzen. Es empfiehlt sich stattdessen, die Lernenden im ersten Schritt lediglich kleine Änderungen am bestehenden Programmcode vornehmen zu lassen. Danach kann man sie dann dazu motivieren, neue Codeteile in bereits bestehenden Quellcode einzufügen. BlueJ greift hierbei sowohl dem Lehrer als auch den Schülern unterstützend unter die Arme.
- Lese Code: Die oft praktizierte Vorgehensweise, dass Schüler zuerst einen Code schreiben, ohne jemals davor einen gelesen zu haben, ist eine merkwürdige Eigenheit des Programmierunterrichts. In anderen Disziplinen wäre dies nur sehr schwierig vorstellbar. So steht z. B. im Deutschunterricht das Lesen von Essays selbstverständlich vor dem Schreiben eigener Texte. Erschwerend kommt dann noch oft hinzu, dass die Unterrichteten überhaupt keinen fremden Quellcode zu lesen bekommen. Das ist schade, denn die Schüler lernen durch das Studieren von gut geschriebenen Quelltexten jede Menge.
- Verwende größere Projekte: Da viele der Vorzüge der objektorientierten Programmierung erst bei größeren Anwendungen zum Vorschein kommen, sollten zur Einführung auch solche verwendet werden. Sind die gewählten Lehrbeispiele zu klein gegriffen, dann lassen sich für die Schüler oftmals die Vorteile dieses Paradigmas nicht erkennen, was Missmut zur Folge hat.
- Beginne nicht mit main-Methode: Jedes Java-Programm erfordert eine main-Methode, die eine Verbindung zwischen Applikation und Betriebssystem herstellt. Zu beachten ist, dass diese Methode rein gar nichts mit der Objektorientierung zu

tun hat bzw. sogar gegen das objektorientierte Denken verstößt. Es ist deshalb nicht sinnvoll, den objektorientierten Unterricht mit der main-Methode zu beginnen. Dazu bedarf es aber einer passenden Programmierumgebung – BlueJ stellt eine solche dar –, die Methodenaufrufe ohne explizite main-Methode unterstützt.

- Verwende nicht "hello world": Da dieses kurze Programm ein gern verwendetes Einführungsbeispiel darstellt, das aber gegen alle vorhin genannten Regeln verstößt, ist es wenig sinnvoll, den objektorientierten Anfangsunterricht damit zu beginnen. Diese Richtlinie lässt sich dahingehend verallgemeinern, dass die Auswahl der Lehrbeispiele sehr sorgfältig vorzunehmen ist.
- Visualisiere Programmstruktur: Bei der Einführung in die objektorientierte Programmierung sind die Klassen und deren Beziehungen untereinander von großer Bedeutung, da sich die Programmstruktur entscheidend auf die Qualität der Lösung auswirkt. Problematisch ist in diesem Zusammenhang, dass viele Entwicklungsumgebungen den Programmaufbau nicht visualisieren können, was aber für die Diskussion mit den Schülern wichtig wäre. BlueJ unterstützt bei diesen Belangen, da es automatisch Klassendiagramme berechnen und anzeigen kann.
- Sei vorsichtig bzgl. Benutzerschnittstelle: Damit die Schüler mit ihrem Programm interagieren können, ist eine entsprechende Benutzerschnittstelle vonnöten. Dazu gibt es drei mögliche Alternativen: die textbasierte Ein- und Ausgabe, eine graphische Benutzeroberfläche und Applets. Alle drei weisen unterschiedliche Vor- und Nachteile auf, die hier aber nicht näher betrachtet werden sollen. Von Lehrerseite ist aber auf alle Fälle zu berücksichtigen, dass es keine dieser drei Benutzerschnittstellen umsonst gibt.

Zu guter Letzt seien noch zwei allgemeine Ratschläge zur effektiveren Gestaltung des Programmierunterrichts gemäß [Prei08] (S. 35) gegeben:

- Lerne aus Fehlern: Hierbei werden die Schüler mit ihren Fehlern im Programmcode konfrontiert. Es erfolgt eine Auflistung der bei der Implementierung gemachten syntaktischen Fehler, die dann im Anschluss daran von den Lernenden zu beheben sind. Diese Vorgehensweise bedingt einen schülerseitigen Lerneffekt.
- Debugging: Hierbei wird den Schülern der Programmablauf Schritt für Schritt vor Augen geführt. Verwendet wird dazu ein spezielles, eigentlich für die Fehlersuche konzipiertes Programmiertool, das Debugger genannt wird und mittels dem man ein Programm schrittweise ausführen kann.

5.4 Programmiersprachen und -umgebungen

Gemäß [Reic05] (S. 9 f.) eignen sich die meisten der verfügbaren Programmiersprachen und -umgebungen nicht für den Einsteigerunterricht, da diese für Experten gedacht sind. Anfänger überfordert dabei in Zusammenhang mit einer Sprache meist der große Sprachumfang, der auf die enorme Zahl an Objekten und Methoden in den Klassenbibliotheken zurückzuführen ist. Bei den Entwicklungsumgebungen sind für Einsteiger deren Bedienung sowie das Zusammenspiel der enthaltenen Werkzeuge kaum nachvollziehbar.

In anderen Schulfächern bzw. Ausbildungsbereichen hat sich die Unterscheidung zwischen professionellen und Lehrwerkzeugen längst etabliert. So stehen z. B. für den Chemieunterricht entsprechend vereinfachte Laboreinrichtungen zur Verfügung und

verwendet man zur Ausbildung von Piloten Flugsimulatoren, deren Schwierigkeitsgrad je nach Ausbildungsstand anpassbar ist. Es ist deshalb auch für den Programmiereinstieg sinnvoll, solche auf die Schülerbedürfnisse zugeschnittenen Systeme zu verwenden.

Es folgen nun im Anschluss die Anforderungen an Sprachen und Umgebungen, die sich für den Einstiegsunterricht in Sachen Programmierung eignen sollen. Danach werden verschiedene solcher Werkzeuge vorgestellt.

5.4.1 Anforderungen an Anfängersprachen und -umgebungen

Damit Entwicklungssysteme in Zusammenhang mit dem Programmiereinstieg in der Schule zum Einsatz gelangen können, sollten sie nach [Reic05] (S. 10 f.) folgende Eigenschaften mitbringen:

- Komplexität reduzieren: Der Zweck eines Programms besteht darin, steuernd auf eine Maschine einzuwirken und diese dadurch zur Durchführung bestimmter Aktionen zu bewegen. Voraussetzung für die Programmierung ist ein Verständnis dieser Maschine. Speziell für Anfänger ist es aber schwierig, ein solches zu entwickeln, weshalb anfangs eine vom Konzept her einfachere Maschine als der Computer vorteilhaft ist.
- Komplexität verstecken: Für den Entwickler ist es unbedeutend, wie eine Maschine bestimmte Aktionen ausführt. Es gilt hierbei das sogenannte Black-Box-Konzept. Die Semantik der Aktionen, also deren Bedeutung, ist aber sehr wohl von Wichtigkeit und sollte deshalb einfach verständlich sein.
- Visualisierung: Die Aktionen und der Zustand der Maschine sowie der Programmablauf sollten graphisch visualisierbar sein. Dies hilft den Schülern nämlich dabei, ein Programm und dessen Strukturen besser zu verstehen.
- Kleiner Sprachumfang: In Bezug auf die Programmiersprache ist für die ersten Versuche ein kleiner, überschaubarer Sprachumfang wünschenswert. Die Sprache sollte außerdem schrittweise erlernbar sein.
- Einfache Programmierumgebung: Damit sich Programmierneulinge auf das Wesentliche, nämlich die Programmlogik, konzentrieren können, sollte der Editor einfach und intuitiv bedienbar sein und ihnen zur Vermeidung von Syntaxfehlern unterstützend unter die Arme greifen.

In Verbindung mit Mikrowelten – es handelt sich dabei um spezielle Programmiersysteme für Anfänger – nennt das oben angeführte Werk noch folgende wünschenswerten Merkmale:

- Die künstliche Welt sollte ansprechend gestaltet sein und den Schülern anschauliche und konkrete Aufgaben bieten.
- Für die Einarbeitung in das System sollte nicht zuviel Zeit erforderlich sein.
- Es sollten damit auch schwierigere Aufgabenstellung bearbeitet werden können.

[Köll99a] beschäftigt sich ausführlich mit den Anforderungen an eine objektorientierte Unterrichtssprache für den Anfängerbereich. Diese lassen sich durch die folgenden Schlüsselworte, die eingehend im Werk behandelt werden, wiedergeben:

- Klare Konzepte
- Reine Objektorientierung
- Sicherheit bzgl. Fehler
- Hohes Level bzgl. Maschinenabstraktion
- Einfaches Objekt- bzw. Ausführungsmodell
- Lesbare Syntax
- Keine Redundanz
- Kleiner Sprachumfang
- Einfacher Wechsel zu anderen Sprachen
- Unterstützung bei Korrektheitszusicherung
- Geeignete Umgebung

Mit den wichtigsten Voraussetzungen, die eine für den objektorientierten Anfangsunterricht geeignete Entwicklungsumgebung mitbringen muss, setzt sich [Köll99b] intensiv auseinander. Jene lauten in Kurzfassung wie folgt:

- Benutzerfreundlichkeit
- Integrierte Werkzeuge
- Unterstützung von Objekten
- Unterstützung bei Codewiederverwendung
- Unterstützung beim Erlernen
- Unterstützung bei Gruppenarbeit
- Verfügbarkeit

5.4.2 Softwaretools zum Einstieg in Objektorientierung

Zum Erlernen der objektorientierten Konzepte wird von vielen Seiten der Einsatz speziell dafür entwickelter Softwaretools empfohlen. Nach [Bena08] lassen sich dabei folgende drei Typen unterscheiden, die anschließend umfassender beschrieben werden:

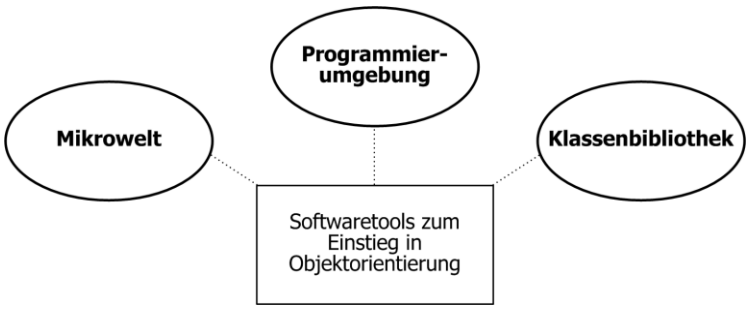


Abbildung 44: Softwaretools zum Einstieg in Objektorientierung

5.4.2.1 Mikrowelt

Bei einer Mikrowelt, oftmals auch als Miniumgebung bezeichnet, handelt es sich in Anlehnung an [Reic05] (S. 9, S. 11 f.) um eine künstliche Programmierwelt, die den Schülern eingeschränkte Möglichkeiten der Programmierung bietet. Sie ist ausschließlich

für Einsteiger gedacht und soll auf die anschließende Verwendung professioneller Sprachen und Umgebungen vorbereiten.

Zumeist wird dabei das Szenario verwendet, dass ein aktives Objekt, wie z. B. ein Marienkäfer, in einer virtuellen Umgebung lebt und mittels entsprechender Befehle gesteuert werden kann. Von den Schülern sind nun Programme zu schreiben, damit dieser Akteur die geforderten Aufgaben in seiner Welt erfüllt.

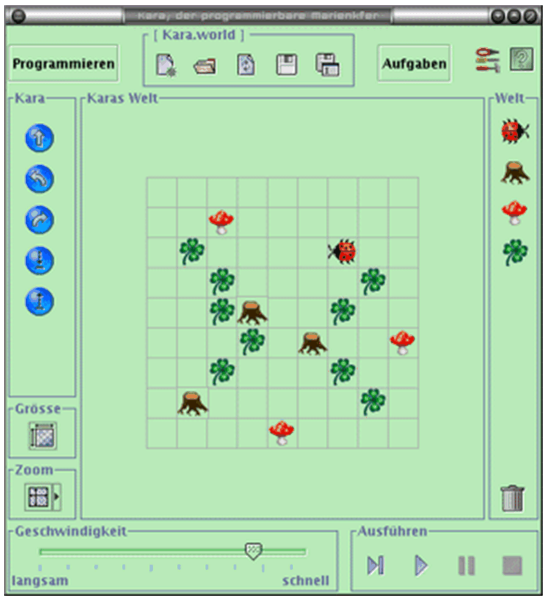


Abbildung 45: Kara, der programmierbare Marienkäfer

Der große Vorteil eines solchen Programmiersystems ist, dass die Lernenden aufgrund des visuellen Programmablaufs ein direktes Feedback bzgl. ihres Programm erhalten und dadurch schnell wahrnehmen können, ob es gemäß der Aufgabenstellung funktioniert oder nicht.

[Altm05] (S. 73) fasst die verschiedenen Eigenschaften wie folgt zusammen:

- Mikrowelten fordern die Kinder zum Aktivwerden heraus.
- Sie unterstützen den natürlichen Lernprozess.
- Sie können abstrakte und schwer zugängliche Welten fassbar machen.
- Sie sind selbst korrigierend.
- Bieten sie einen für Schüler interessanten Hintergrund, können sie sehr motivierend wirken.

Als Beispiel für eine solche Mikrowelt lässt sich Kara nennen, das auch in einer objektorientierten Variante erhältlich ist. Viele weitere solcher Miniumgebungen listen [Frei02] und [Reic05] (S. 133 ff.) auf und beschreiben diese auch überblicksartig.

5.4.2.2 Programmierumgebung

Laut [Börs07] erfolgt die Auswahl der Programmiersprache und -umgebung für den Unterricht oftmals aufgrund der industriellen Verbreitung. Die didaktischen Aspekte werden dabei aber vollkommen außer Acht gelassen. Das hat zur Folge, dass sich die Schüler mit einer komplexen Syntax und Semantik der Sprache, umfangreichen

Bibliotheken und vielen Funktionalitäten der Entwicklungsumgebung herumschlagen müssen, was unweigerlich eine hohe kognitive Belastung zur Folge hat.

Um dem entgegenzuwirken, wurden eigene Programmierumgebungen entwickelt, die Anfängern einen einfacheren Einstieg ermöglichen. Solche Systeme sind zwar bei weitem nicht so leistungsfähig wie professionelle Werkzeuge, durch eine sinnvolle Reduktion der Konzepte und Funktionalitäten auf das Wesentliche aber deutlich einfacher und schneller für Programmierneulinge zu erlernen und einzusetzen. Des Weiteren enthalten sie meist auch hilfreiche Lehrwerkzeuge, die den Schülern z. B. die Visualisierung der Klassenstruktur erlauben.

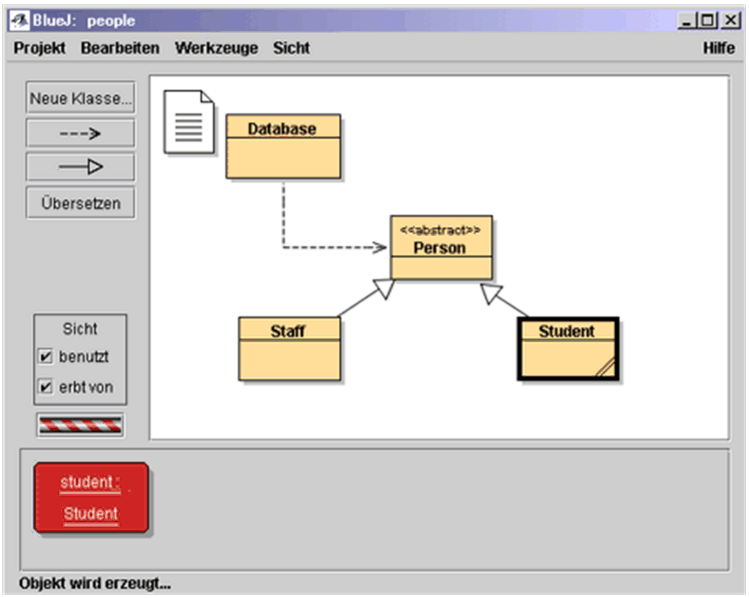


Abbildung 46: BlueJ

Als Beispiel für eine solche, speziell für den Ausbildungsbereich entwickelte Entwicklungsumgebung ist BlueJ zu nennen. Dieses System verfolgt das Ziel, Programmieranfängern den Einstieg in die objektorientierte Programmierung mit Java deutlich zu vereinfachen.

5.4.2.3 Klassenbibliothek

Klassenbibliotheken stellen eine weitere Möglichkeit zur schülergerechten Einführung in eine Programmiersprache und deren Konzepte dar. Zu beachten ist, dass diese nicht eigenständig lauffähig sind, sondern eine passende Umgebung erfordern.

Bei diesem Ansatz wird versucht, durch ein geschicktes Klassendesign jene Aspekte besonders hervorzuheben, die im Rahmen des Unterrichts den Schülern vermittelt werden sollen. Gleichzeitig wird dabei aber auch das Ziel verfolgt, die Komplexität durch bewusstes Verstecken von Details zu reduzieren, um den Lernenden dadurch den Beginn zu erleichtern.

Als Beispiel für eine zur Einführung in Java gedachte Klassenbibliothek kann Objectdraw genannt werden.

6 Konkrete Ansätze zur Vermittlung objektorientierter Konzepte

Dieses Kapitel stellt verschiedene Ansätze zur Einführung in die Objektorientierung und deren zugehörige Konzepte auf Schülerniveau vor. Es wird hierbei eine grobe Kategorisierung dieser Vorgehensweisen nach unterschiedlichen Gesichtspunkten vorgenommen.

6.1 Objektspiele

Hierbei werden die internen Abläufe eines objektorientierten Systems von den Unterrichteten in Form eines Rollenspiels durchgespielt. Die Schüler lernen dabei die Objekte als gekapselte, autonome Akteure kennen, die durch Nachrichtenaustausch mit anderen Objekten interagieren können. Durch die handlungsorientierte Vorgehensweise – jeder Lernende ist hierbei ein integraler Teil des Gesamtsystems und bringt sich mit vielen seiner Sinne in das Spiel ein – ist ein guter und langfristig anhaltender Lerneffekt seitens der Schüler zu erwarten.

6.1.1 Bankautomat

Bei diesem Ansatz gemäß [Dißm03] werden die Schüler anhand eines alltäglichen Beispiels aus dem Bankbereich in die objektorientierten Konzepte eingeführt. Im Konkreten spielen jene mittels eines Rollenspiels die Verwendung und die Funktionalität eines Bankautomaten nach, der zur Abfrage von Kontoständen und zur Behebung von Bargeld genutzt werden kann.

Der Lehrer erklärt eingangs die vier Grundregeln, die für jeden Mitspieler während des Spielens gelten und die dieser verpflichtend einzuhalten hat:

- Er hält sich an die ihm vorgegebenen Handlungsanweisungen.
- Er fragt andere Spezialisten, sobald er an die Grenzen seiner Handlungsmöglichkeiten stößt.
- Er hilft anderen Spezialisten, sofern ihn diese um Hilfe bitten.
- Er sichert seinen Status als Spezialist dadurch, dass er die ihm zugeordneten Daten und Handlungsanweisungen streng vertraulich behandelt.

Danach beschreibt er die Aufgabe des Gesamtsystems und weist seinen Schülern die einzelnen Rollen zu. Während das Spiel läuft, kommt ihm dann die Aufgabe des überwachenden Organs zu.

Von den Schülern sind verschiedene Rollen zu übernehmen und zu spielen. Und zwar gibt es jeweils eine Spezialisten für den Kundenkontakt, die Kontenverwaltung, die Sollzinsberechnung und die Banknotenausgabe und mehrere, jeweils ein Konto repräsentierende Experten. Dazu kommen noch die Kunden, die den Bankautomaten nutzen. Zu jeder Rolle gibt es eine vertraulich zu behandelnde Kurzbeschreibung, die den Rollenakteur über seine Daten und Handlungsanweisungen informiert, gemäß denen er zu agieren hat.

Spezialist für Kontenverwaltung

Aufgaben
Verwalten der Zuordnung von Kontonummer, Kartennummer, PIN und Kontobezeichnung.
Diese Zuordnung darf nicht verändert werden.
Die Zuordnung ist sicherheitskritisch und darf nicht herausgegeben werden.

Kontonummer	Kartennummer	PIN	Kontobezeichnung
2341	1432	4567	6
5109	9015	3128	4
1344	4413	6891	3
...	...	...	...

Abbildung 47: Kurzbeschreibung für Kontenverwaltungsrolle

Der Austausch von Informationen zwischen den verschiedenen Spezialisten erfolgt ausschließlich mittels schriftlicher Kurznachrichten, die dem Frage/Antwort-Prinzip folgen.

Es wird mit einfachen Aufgaben, wie z. B. dem Abfragen des Kontostandes eines existierenden Kontos, begonnen. Mit der Zeit werden die Übungen immer schwieriger und herausfordernder. So sollen die Schüler bewusst auch mit Ausnahmesituationen, wie z. B. einer falschen persönlichen Identifikationsnummer (PIN) oder einem nicht existierendem Konto, konfrontiert werden.

Die ersten Schritte verlaufen dabei bei jeder Aufgabenstellung gleich. Ein Kunde übergibt dem Spezialisten für Kundenkontakt seine Anforderung in schriftlicher Form. Dieser überprüft die Angaben und reicht jene im Korrektheitsfall an den Spezialisten für Kontenverwaltung weiter. In Abhängigkeit der Kundenanforderung führt dieser dann bestimmte Aktionen durch bzw. veranlasst deren Durchführung.

Nach einer ausreichenden Spielzeit arbeitet dann der Lehrer gemeinsam mit seinen Schülern die objektorientierten Gestaltungsprinzipien heraus. Anhand der unterschiedlichen Konto-Objekte lässt sich zusätzlich zum Objektbegriff auch der Klassenbegriff einführen. Das Vererbungsprinzip kann durch die Verwendung unterschiedlicher Kontoarten hergeleitet werden.

6.1.2 Mensch ärgere dich nicht

[Diet05a] beschreibt ebenfalls eine Methode, um den Schülern die objektorientierten Grundkonzepte näherzubringen. Die dem Ganzen zugrundeliegende Aufgabe ist die Erstellung eines Programms zum Spielen von "Mensch ärgere dich nicht". Der Programmablauf wird dabei von den Lernenden mittels eines entsprechenden Objektspiels erarbeitet.

Von den Schülern, denen noch zusätzlich die Augen verbunden werden, sind im Rahmen des Rollenspiels die folgenden Objekte darzustellen: Spieler, Spielstein, Würfel sowie mehrere Spielfelder. Zwecks besserer Überschaubarkeit wird lediglich ein Spieler – logischerweise dann auch nur ein Spielstein – verwendet. Jedes Spielfeld wird von genau einem Lernenden repräsentiert. Um die Verbindung zwischen den einzelnen

Feldern zu symbolisieren, legt jeder Vorgänger seinem Nachfolger den Arm auf die Schulter. Aus technischer Sicht entspricht dies dem Aufbau einer verketteten Liste.

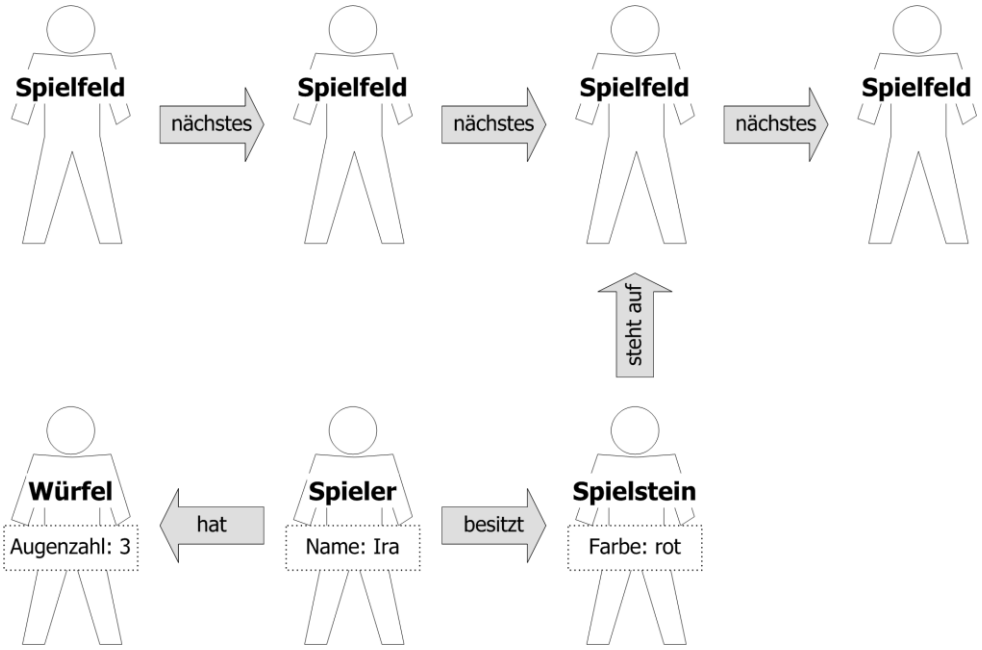


Abbildung 48: Schülerrollen und Beziehungen zwischen diesen

Gemäß den Spielregeln von "Mensch ärgere dich nicht" muss der Spielstein des Spielers um die der gewürfelten Augenzahl entsprechende Anzahl an Spielfeldern weiterbewegt werden. Beim Rollenspiel merkt der Spielstein-Schüler rasch, dass er die Würfelzahl gar nicht kennt und deshalb diese über den Spieler-Schüler – er ist der Einzige, der mit dem Würfel-Schüler interagieren kann – erfragen muss. Die Augenzahl merkt sich der Spielstein-Schüler durch entsprechendes Mitzeigen mit seinen Fingern. Im Anschluss daran fordert der Spielstein-Schüler den ihm aktuellen zugeteilten Spielfeld-Schüler zur Nennung seines Nachfolgers auf, damit er seinen Arm auf jenen vorrücken kann. Diese Vorgehensweise – also Nachfolgersuche und Vorrückung – erfolgt so oft, bis die Anzahl an Wiederholungen mit der vom Spielstein-Schüler gemerkten Würfelzahl übereinstimmt.

Die am Rollenspiel beteiligten Schüler teilen ihre Aktionen den unbeteiligten Lernenden mit. Zweitere halten deren Kommentare auf der Tafel fest, wodurch eine textuelle Beschreibung des Programmablaufes entsteht.

Durch diesen Ansatz wird für die Schüler der Begriff des Objektes greifbar. Vor allem erkennen sie durch das Rollenspiel, welchen Einschränkungen ein Objekt bei der objektorientierten Programmierung unterliegt und wie durch das Zusammenwirken mehrerer Objekte die gewünschte Programmfunktionalität umgesetzt werden kann.

6.2 Modellierung

Die Einführung in die objektorientierten Begrifflichkeiten erfolgt hierbei mittels der objektorientierten Modellierung. Anhand eines aus dem Leben gegriffenen Beispiels werden die entsprechenden Konzepte schrittweise eingeführt und liegt am Ende dann ein Modell der realen Gegebenheiten vor. Oftmals folgt dann nach der Modellierungsphase noch die Umsetzung des Modells in ein lauffähiges Programm. Ob

der letzte Schritt allerdings im Rahmen einer Einführung überhaupt sinnvoll ist, darüber lässt sich sicherlich diskutieren.

6.2.1 Eisenbahnverkehr

Beim Ansatz gemäß [Hubw07] (S. 209 ff.) werden die Schüler anhand eines Beispiels, das dem Bereich des Eisenbahnverkehrs entstammt, in die objektorientierten Basiskonzepte eingeführt. Es wird dabei der Ausschnitt einer Gleisanlage betrachtet, für den von den Lernenden der Verkehr von drei Zügen zu simulieren ist. Die Steuerung aller beteiligten Objekte erfolgt nicht durch ein zentrales Steuerwerk, sondern jedes Objekt bestimmt durch Kommunikation mit den anderen seine nächste Aktion.

Im ersten Schritt wird der Gleisplan gezeichnet, der folgendes Aussehen hat:

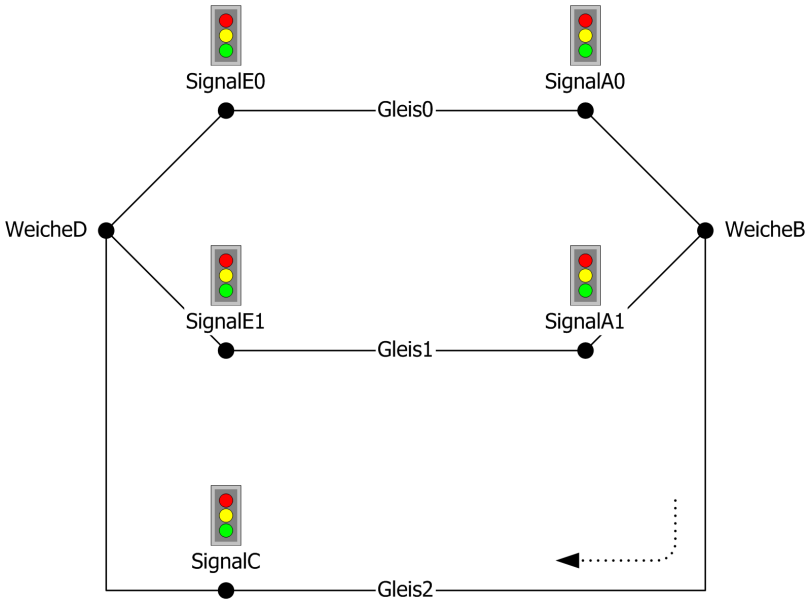


Abbildung 49: Gleisplan zum Eisenbahnverkehr

Danach werden dann die Rand- und Anfangsbedingungen erarbeitet und schriftlich festgehalten. Zwecks Vereinfachung wird anfangs jegliche Nebenläufigkeit ausgeschlossen. Somit kann zur gleichen Zeit immer nur ein Objekt agieren und hat dabei stets exklusiven Zugriff auf alle anderen Objekte.

Anschließend wird gemeinsam für einen der Züge ein vollständiger Umlauf durchgespielt und die Vorgehensweise in textueller Form notiert. Nachdem auch noch die Steuerbedingungen für den Gesamtablauf festgehalten wurden, wodurch die Einzelaktivitäten der Züge miteinander verknüpft werden, kann mit der eigentlichen Modellierung begonnen werden.

Die Objekte lassen sich in diesem Beispiel recht einfach identifizieren und können auch gleich in entsprechende Klassen zusammengefasst werden:

- Züge: Zug1, Zug2, Zug3
- Signale: SignalA0, SignalA1, SignalC, SignalE0, SignalE1
- Weichen: WeicheB, WeicheD

Die Attribute und Methoden dieser Objekte ergeben sich aus entsprechenden, pragmatischen Überlegungen. Schlussendlich gelangt man von den einzelnen Objekten zu den folgenden Klassen:

	Attribute	Methoden	Objekte
Zug	• Nummer • Gleis • Position	• Position mitteilen • Vorrücken	• Zug1 • Zug2 • Zug3
Signal	• Zustand	• Zustand mitteilen • Zustand setzen	• SignalA0 • SignalA1 • SignalC • SignalE0 • SignalE1
Weiche	• Zustand	• Zustand mitteilen • Zustand setzen	• WeicheB • WeicheD

Tabelle 8: Klassen mit zugehörigen Attributen, Methoden und Objekten

Anhand der offensichtlichen Gemeinsamkeiten zwischen den Klassen Signal und Weiche lässt sich nun auch noch der Vererbungsbegriff einführen. Und zwar werden die beiden von der Oberklasse Steuerelement abgeleitet. Das vollständige Klassendiagramm hat dann folgendes Aussehen:

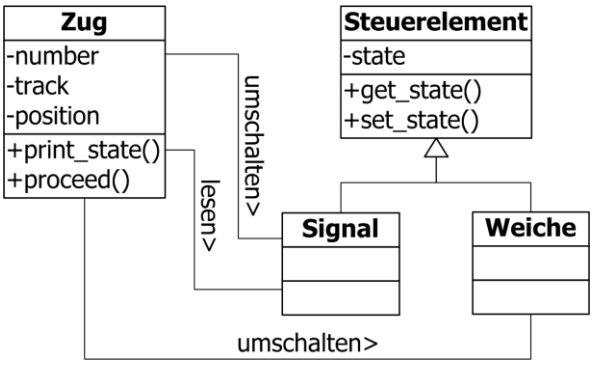


Abbildung 50: Klassendiagramm für Eisenbahnverkehr

Im Objektmodell fehlt nun noch der zeitliche Ablauf der Methoden. Dieser wird mittels mehrerer Zustandsübergangsdiagramme modelliert. Für die Steuerelemente ist dies noch relativ einfach zu bewerkstelligen, in Verbindung mit den Zügen wird dies allerdings für die Schüler eine kleine Herausforderung werden.

Zu guter Letzt erfolgt die Umsetzung des erarbeiteten Modells in ein einfaches lauffähiges Programm. Verwendet wird dazu die Programmiersprache Java. Zu erwarten ist allerdings, dass dieser Schritt speziell für Einsteiger mit großen Schwierigkeiten verbunden sein wird.

6.2.2 Bibliothek

Der Ansatz unter [Schu04] (S. 195 ff.) ist dem vorherigen sehr ähnlich. Als Szenario wird hier nun eine Bibliothek verwendet, die es Lesern ermöglicht, unterschiedliche Bücher auszuleihen und natürlich auch wieder zurückgeben.

Im ersten Schritt ist lediglich ein einfaches Programm zu entwickeln, das den Lieblingsautor und das Lieblingsbuch als Eingabe entgegennimmt und daraufhin eine Ausgabe für das persönliche Buch des Monats generiert. Nach einer gemeinsamen Analysephase werden die beiden identifizierten Klassen Autor und Buch mittels eines Klassendiagramms modelliert und der Programmablauf in Form eines Struktogramms beschrieben. Auf das selbstständige Programmieren wird eingangs verzichtet. Stattdessen erhalten die Schüler den fertigen, in Python geschriebenen Programmcode zum Lernen und Herumprobieren vorgelegt.

Nach diesen ersten Modellierungsversuchen wird nun das eigentliche Programm entworfen und umgesetzt. Schrittweise werden anhand des Bibliothekszenarios die Konzepte des Objektes, der Klasse und der Vererbung eingeführt und erklärt. Schließlich gelangt man dann zu dem folgenden Klassendiagramm:

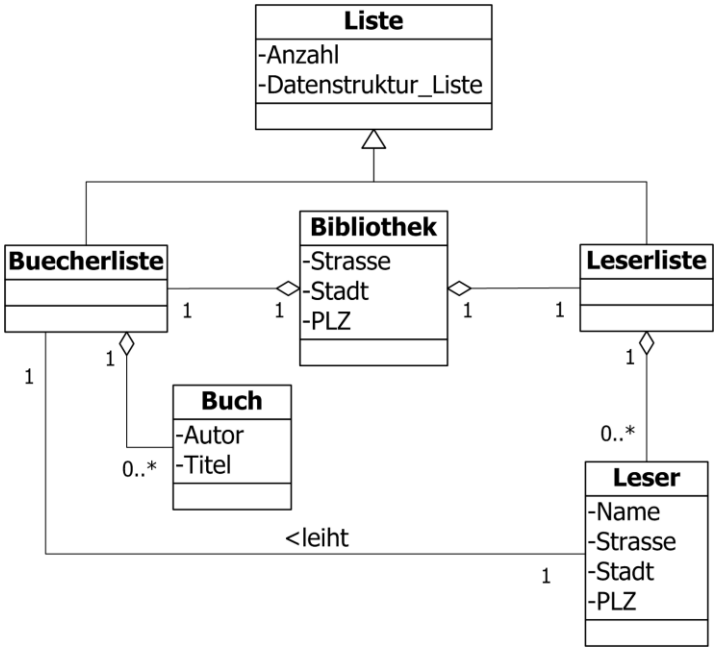


Abbildung 51: Klassendiagramm für Bibliothek

Nun gilt es noch die dynamischen Abläufe im Programm zu erarbeiten und zu modellieren. Hierzu wird eingangs analysiert, welche Botschaften die verschiedenen Objekte untereinander austauschen müssen, um bestimmte Aktionen, wie z. B. das Zurückgeben eines geliehenen Buches, kooperierend durchzuführen. Zur Darstellung der dynamischen Abläufe eignen sich Sequenzdiagramme sehr gut, weshalb deren Verwendung empfohlen wird.

In der letzten Phase kann dann wiederum das gemeinsam erarbeitete Modell unter Verwendung der Sprache Python in ein Programm umgesetzt werden.

6.3 Mikrowelten

Hierbei ist von den Schülern das Verhalten eines virtuellen Akteurs derart zu beeinflussen, so dass dieser bestimmte Aufgaben in seiner Mikrowelt erfüllt. Da es sich bei solchen Mikrowelten um reine Lernumgebungen handelt, können bei deren Gestaltung die didaktischen Aspekte im Vordergrund stehen. Klarerweise wirkt sich das auf die Vermittlung der zu lehrenden Konzepte positiv auswirkt.

6.3.1 Java-Hamster-Modell

Bei diesem Ansatz gemäß [Bole04] und [Bole05] sind von den Schülern in einer mit Java fast deckungsgleichen Sprache sogenannte Hamster-Programme zu schreiben, mittels derer virtuelle Hamster durch eine virtuelle Landschaft gesteuert werden und dabei bestimmte Aufgaben erfüllen. Im Rahmen der Übungsaufgaben werden Schritt für Schritt die wichtigsten Programmierkonzepte eingeführt, wobei natürlich die Einführung der objektorientierten Prinzipien in Zusammenhang mit dieser Arbeit von besonderer Bedeutung ist.

Es gilt dabei das folgende Szenario: Die Hamster leben in einer durch Kacheln repräsentierten Landschaft und können sich in dieser in alle vier Himmelsrichtungen bewegen. Auf jeder Kachel können dabei ein oder mehrere Körner liegen, die von den Hamstern aufgesammelt werden können. Es gibt aber auch Mauerkacheln, welche die Hamster in ihrer Vorwärtsbewegung blockieren. Jeder Hamster kann beliebig viele Körner im Maul haben und solche auch wieder auf die Kacheln zurücklegen.

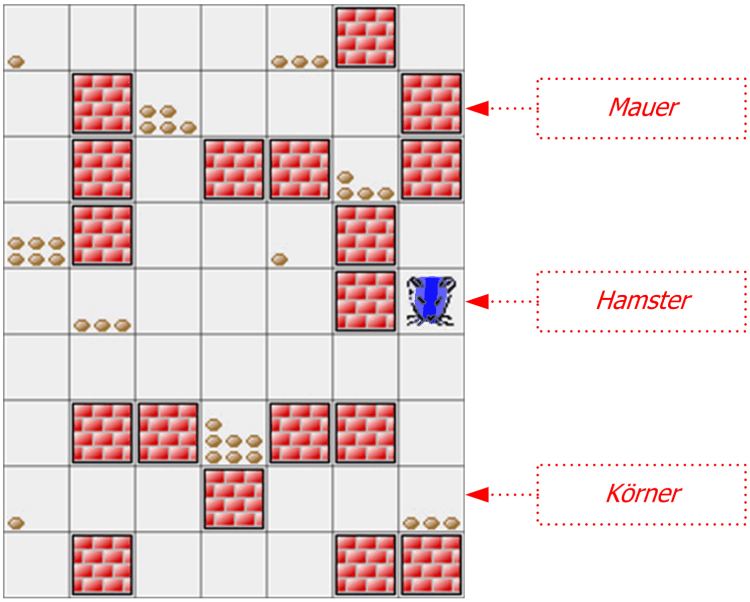


Abbildung 52: Landschaftsaufbau beim Java-Hamster-Modell

Zur Steuerung eines Hamsters kann sich der Lernende folgender Befehle bedienen:

- vor: Bewege den Hamster in Blickrichtung um eine Kachel nach vorne.
- linksUm: Drehe den Hamster um 90 Grad nach links.
- nimm: Sammle ein Korn von der Kachel auf, auf der sich der Hamster gerade befindet.
- gib: Lege ein Korn auf der Kachel ab, auf der sich der Hamster gerade befindet.
- vornFrei: Überprüfe, ob die nächste Kachel in Blickrichtung des Hamsters frei ist.
- kornDa: Überprüfe, ob ein Korn auf der Kachel liegt, auf der sich der Hamster gerade befindet.
- maulLeer: Überprüfe, ob der Hamster mindestens ein Korn im Maul hat.

Zur Programmerstellung und -ausführung stellt der Java-Hamster-Simulator jede Menge nützliche Werkzeuge bereit: einen Editor zum Programmieren, einen Compiler,

Interpreter und Debugger zum Übersetzen, Ausführen und Testen des Programms sowie einen Territoriumsgestalter zur Landschaftserstellung.

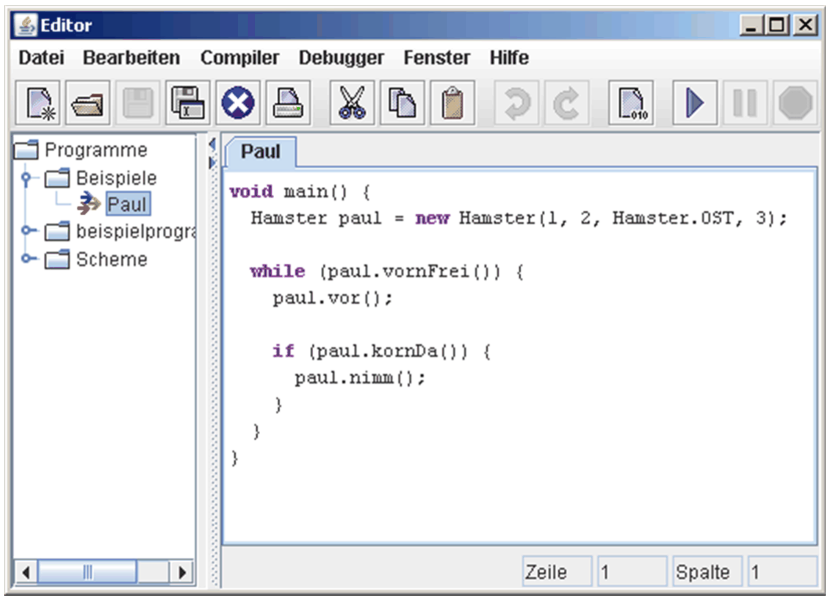


Abbildung 53: Java-Hamster-Simulator

Anfangs werden die Lernenden schrittweise in die grundlegenden Programmierkonzepte, wie z. B. Anweisungen, Funktionen, Kontrollstrukturen, Variablen oder Rekursionen, eingeführt. Darauf aufbauend erfolgt dann im nächsten Schritt die Vermittlung der grundlegenden, objektorientierten Konzepte. Die Schüler lernen nun, wie man unter Verwendung der vordefinierten Klasse Hamster neue Hamster-Objekte dynamisch anlegen kann und lernen dabei den Unterschied zwischen dem Klassen- und dem Objektbegriff kennen.

Eine der ersten Übungsaufgaben ist das Erstellen eines Programms, bei dem der Hamster Paul beim Programmstart in der ersten Reihe und zweiten Spalte, mit Blickrichtung Osten und drei Körnern im Maul positioniert wird und sich dann solange vorwärts bewegt bis ihn eine Mauer blockiert. Auf jeder Kachel, die er betritt, soll er ein Korn aufsammeln, sofern zumindest eines darauf liegt. Der Quelltext zu diesem Beispiel lautet wie folgt:

```
void main() {
  Hamster paul = new Hamster(1, 2, Hamster.OST, 3);

  while (paul.vornFrei()) {
    paul.vor();

    if (paul.kornDa()) {
      paul.nimm();
    }
  }
}
```

Das Prinzip der Vererbung wird dann anschließend durch die Ableitung spezifischerer Klassen von der Basisklasse Hamster demonstriert. Für den Einstieg in die Vererbung eignet sich dazu das folgende Beispiel eines erweiterten Hamsters, der sich nicht nur nach links, sondern auch um 180 Grad sowie nach rechts drehen kann:

```
class DrehHamster extends Hamster {
  void kehrt() {
    linksUm();
    linksUm();
  }

  void rechtsUm() {
    kehrt();
    linksUm();
  }
}
```

Übungsbeispiele, die den Lernenden das Prinzip des Polymorphismus näherbringen, werden ebenfalls angeführt. Zusätzlich sieht der Ansatz auch eine Einführung in die parallele Programmierung vor. Hierzu wird den Schülern das Thread-Konzept in Java vorgestellt, das die eigenständige Koordination der Hamster untereinander möglich macht.

Sehr große Ähnlichkeit mit der soeben vorgestellten Herangehensweise weist jene unter [Beck01] auf. Sie basiert allerdings auf der Idee von Karel the Robot.

6.3.2 The Object Shop

Dieser Ansatz nach [Wood97] bedient sich einer auf Compact Disc Read-Only Memory (CD-ROM) bereitgestellten multimedialen Lernumgebung, um Programmierneulinge in die Kernprinzipien der objektorientierten Programmierung einzuführen. Die Grundidee hierbei ist, dass die Schüler ein grundlegendes Verständnis für die Konzepte entwickeln, bevor sie sich mit dem Schreiben von Programmcode und der Syntax der zugehörigen Sprache beschäftigen.

Als Szenario gelangt hierbei ein vierstöckiges Kaufhaus in dreidimensionaler Darstellung zum Einsatz, in dem von den Lernenden verschiedene, virtuelle Einkäufe durchzuführen sind. Zur Bewerkstelligung solcher Käufe muss der Schüler mit verschiedenen Objekten, wie z. B. Geldkassetten, Kreditkarten, Kalendern, diversen Behältnissen oder Kaufartikeln unterschiedlicher Klassen, interagieren.

Die Interaktion an sich erfolgt über sogenannte Objektcontroller. Es handelt sich dabei um Dialogfenster, die sich beim Anklicken eines Objektes öffnen und über die Nachrichten an dieses zwecks Durchführung bestimmter Aktionen geschickt werden können. Weiters können darüber die Objekteigenschaften abgefragt werden.

Abbildung 54: Objektcontroller für Hund-Objekt

Je höher man stockwerksmäßig im Kaufhaus nach oben steigt, desto umfangreicher werden die Möglichkeiten, die man in Zusammenhang mit den verfügbaren Objekten anstellen kann. Auch der Grad der Formalisierung steigt mit jedem Stockwerk. Weiters werden dem Lernenden zusätzliche Werkzeuge zur Verfügung gestellt.

6.4 Programmierumgebungen

Die Vermittlung der objektorientierten Konzepte erfolgt hierbei mittels entsprechender Programmierumgebungen bzw. mittels spezieller Erweiterungen, die sich gängiger Systeme als Basis bedienen. Sie sind vorrangig nach didaktischen Gesichtspunkten gestaltet und ermöglichen dadurch Anfängern einen einfachen und schnellen Einstieg in die Programmierung. Die Leistungsfähigkeit und Funktionsvielfalt steht bei solchen Systemen im Gegensatz zu professionellen Entwicklungsumgebungen nicht im Vordergrund. Eine entsprechende Reduktion auf das Wesentliche ist für Lernzwecke auch sehr empfehlenswert. Eine graphische Unterstützung für die einzelnen Arbeitsschritte sowie umfassende Visualisierungsmöglichkeiten sind natürlich ebenfalls wünschenswert.

6.4.1 BlueJ

[Barn06] stellt einen Ansatz vor, wie man Programmierneulinge mittels BlueJ in die objektorientierte Programmierung mit Java einführen kann. Dieser wird in Anschluss an die Kurzvorstellung dieser Programmierumgebung näher beschrieben.

Bei BlueJ handelt es sich um eine interaktive Java-Entwicklungsumgebung, die speziell auf Anfänger abzielt und jene beim Erlernen der objektorientierten Programmierung sowie der zugrundeliegenden Konzepte unterstützen soll. Dem Anwender steht in BlueJ eine Vielzahl an graphischen Werkzeugen zur Verfügung, wodurch sich für diesen der Einstieg deutlich vereinfacht. Die Tools ermöglichen nicht nur die teilweise Generierung von Quelltext, sondern bieten dem Benutzer durch entsprechende Visualisierung auch unterschiedliche Sichten auf das Programm bzw. auf Teile davon.

Im ersten Schritt des Ansatzes werden anhand eines Figuren-Beispiels die Begriffe der Klasse und des Objektes eingeführt und den Lernenden der Unterschied zwischen diesen beiden objektorientierten Konzepten vor Augen geführt. Die Klassen und Objekte eines Projektes werden in BlueJ im Hauptfenster angezeigt. Das Anlegen neuer Klassen, aber auch das Erzeugen von Objekten dieser Klassen kann natürlich ebenfalls auf graphischem Wege erfolgen.

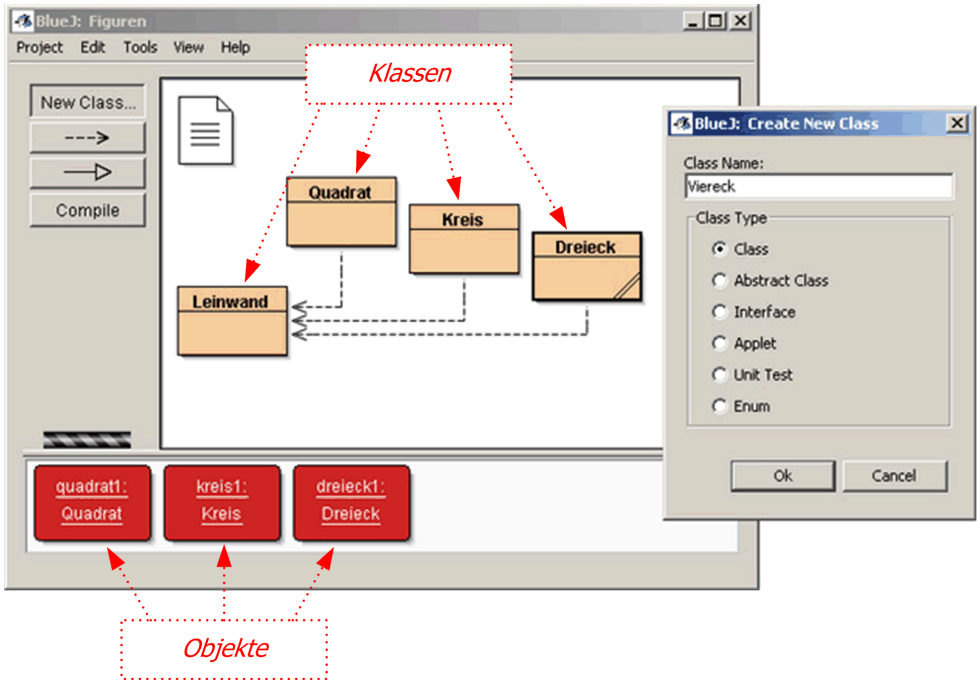


Abbildung 55: Klassen und Objekte in BlueJ

Auch die Methoden eines Objektes sind in BlueJ derart aufrufbar, wobei die Eingabe der Methodenparameter über ein Dialogfenster vonstatten geht. Als Parameter können dabei nicht nur einfache Datentypen, sondern auch Objekte übergeben werden. Mittels des Objektinspektors kann jederzeit ein Blick auf die Werte eines Objektes geworfen werden.

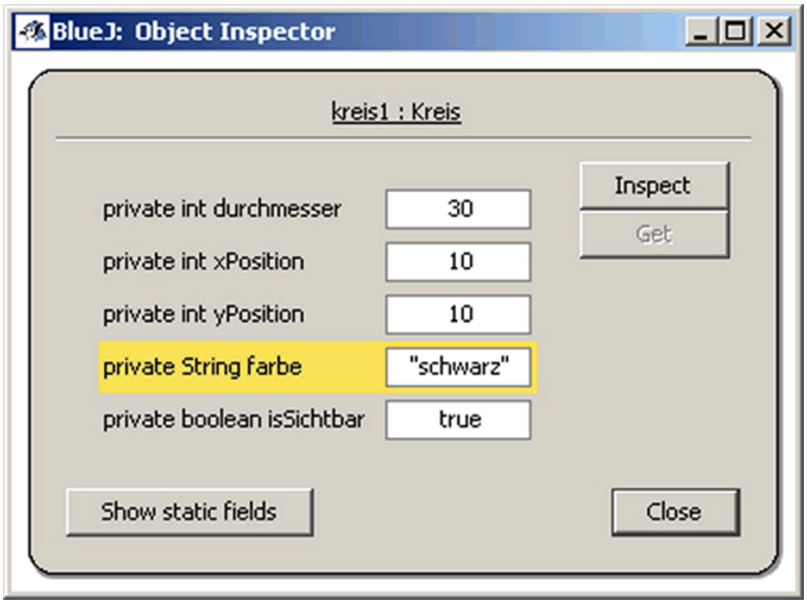


Abbildung 56: Objektinspektor für Kreis-Objekt

Durch das Beispiel eines Ticketautomaten wird dann im nächsten Abschnitt das Thema Klassendefinition vertieft. Dazu werden der dem Programm zugrundeliegende, bestehende Java-Quelltext eingesehen und den Lernenden anhand jenes der Sinn und die Verwendung von Datenfeldern, Konstruktoren und Methoden nähergebracht. Die Schüler lernen hierbei nun das Prinzip der Kapselung kennen.

Die Interaktion von Objekten wird dann anschließend mittels eines Uhren-Beispiels – es wird die Anzeige einer Digitaluhr modelliert – demonstriert. Die Zusammenhänge zwischen den Klassen und Objekten können hierbei durch Klassendiagramme für die statische und durch Objektdiagramme für die dynamische Sicht dargestellt werden.

Anhand weiterer, zunehmend komplexer werdender Beispiele erfolgt dann im Anschluss daran eine Einführung in die Verwendung von Objektsammlungen und die Nutzung von Bibliotheken. Des Weiteren werden die Themenkreise der Fehlervermeidung und des Klassenentwurfs angerissen.

Das Vererbungsprinzip ist dann das Thema des nächsten Abschnitts und wird mittels eines Programmbeispiels zur Verwaltung unterschiedlicher Datenträger erarbeitet. Bei ihnen kann es sich um Compact Discs (CD) oder Digital Versatile Discs (DVD) handeln. Wenig überraschend weisen diese beiden Typen viele Gemeinsamkeiten auf und bietet sich hier die Anwendung der Vererbung an. Es wird deshalb die Oberklasse Medium eingeführt, von der die Klassen CD und DVD abgeleitet werden. Dies lässt sich in BlueJ auf graphische Art und Weise bewerkstelligen. Die Vererbungsbeziehungen zwischen der Oberklasse und ihren abgeleiteten Klassen werden dann natürlich ebenfalls entsprechend dargestellt.

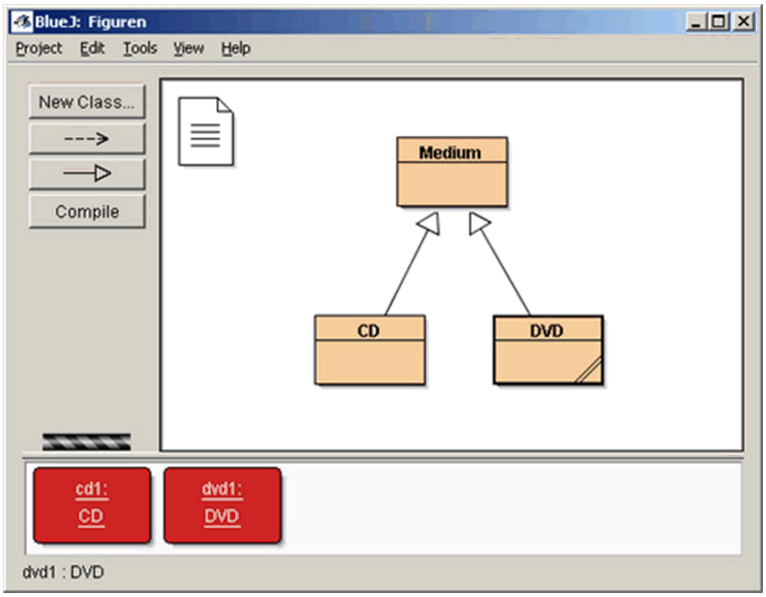


Abbildung 57: Vererbungsbeziehungen in BlueJ

Durch einen Blick auf den zugehörigen Java-Quelltext lernen die Schüler, mittels welcher Sprachkonstrukte die Vererbung in dieser Programmiersprache realisiert wird.

In Folge werden noch speziellere Kapitel der Vererbung, wie z. B. das Subtyping oder der Cast-Operator, behandelt. Der Polymorphismus wird dann im Anschluss daran

anhand des Überschreibens von Methoden demonstriert. Auch abstrakte Klassen und das Interface-Konzept von Java kommen zur Sprache.

In Zusammenhang mit diesen Themen lässt sich allerdings festhalten, dass sie weit über die Lehrziele des Schulunterrichts hinausgehen und didaktische Vorteile durch die Verwendung von BlueJ bei der Vermittlung dieser nicht feststellbar sind, weshalb darauf nicht näher eingegangen wird.

6.4.2 Fujaba life^3

Bei diesem in drei Phasen ablaufendem Ansatz nach [Meye04] zur Einführung von Schülern in die objektorientierten Konzepte gelangt Fujaba life^3 zum Einsatz.

Es handelt sich dabei um ein auf Fujaba aufsetzendes CASE-Tool, das speziell für den Schuleinsatz entwickelt worden ist. Es unterstützt die Erstellung von Klassen- und Storydiagrammen. Zweitere stellen eine Kombination aus Aktivitäts- und erweiterten Kollaborationsdiagrammen dar und erlauben das Erstellen vollständiger Methoden, ohne dass der Lernende dafür auch nur eine einzige Zeile eigenen Code schreiben muss. Mittels eines Plugins lässt sich Fujaba life^3 in die weit verbreitete Programmierumgebung Eclipse integrieren. Aus den Diagrammen kann dort dann auf Knopfdruck der zugehörige Java-Quellcode generiert und als Eclipse-Projekt exportiert werden. Zur Ausführung bzw. Nachbearbeitung des Quelltextes steht dem Benutzer natürlich die volle Funktionalität von Eclipse zur Verfügung.

Im ersten Schritt des Ansatzes geht es darum, dass die Schüler eine Verständnis für Objekte, deren Strukturen und das Zusammenwirken zwischen diesen entwickeln. Um dieses Ziel zu erreichen, erzeugen die Schüler unter Anleitung des Lehrers Instanzen von den bereitgestellten, vorgefertigten Klassen und bauen mittels der Objekte ein erstes, lauffähiges Programm zusammen. Um die Objektstrukturen während der Laufzeit zu untersuchen aber auch zu manipulieren, wird der dynamische Objektbrowser verwendet. Dieses Fujaba-Werkzeug erlaubt den Lernenden ein Herumprobieren und -spielen mit ihren Objekten.

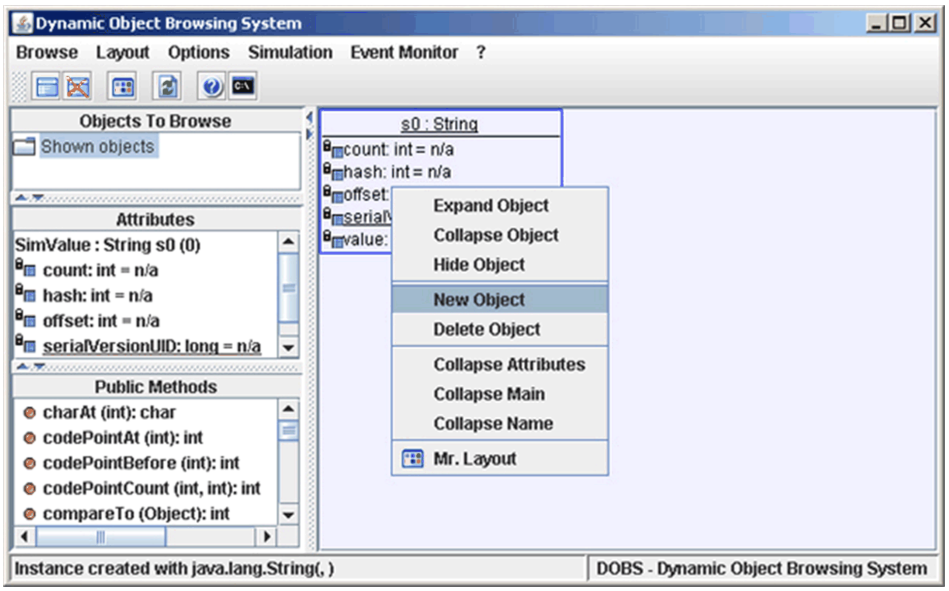


Abbildung 58: Dynamischer Objektbrowser in Fujaba life^3

In der zweiten Phase werden den Schülern der Klassenbegriff samt Attributen und Methoden sowie die Beziehungen zwischen den Klassen nähergebracht. Auch hier erarbeitet die Schulklasse gemeinsam mit dem Lehrer den Lernstoff. Es gelangen dabei nun Klassendiagramme und Storydiagramme zum Einsatz.

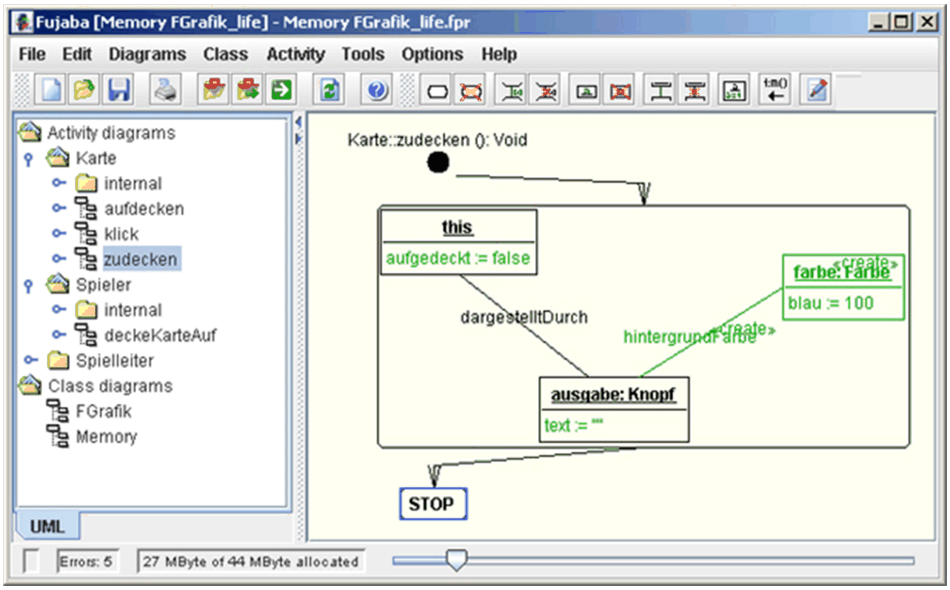


Abbildung 59: Storydiagramm in Fujaba life^3

Mittels dieser Konstrukte können die Lernenden in Fujaba life^3 vollständige Applikationen bauen, ohne dass sie sich dabei mit der Syntax einer Programmiersprache herumschlagen müssen. Das ermöglicht ihnen, sich voll auf das Erlernen der objektorientierten Konzepte zu konzentrieren. Zum Zwecke des Kennenlernens einer textuellen Programmiersprache wird im Anschluss daran der Java-Quellcode aus den visuellen Klassen- und Methodendarstellungen generiert. Die Schüler können nun die graphischen mit den zugehörigen textuellen Konstrukten vergleichen und so den Einstieg in die objektorientierte Programmierung mit Java angemessen vollziehen.

Um die in den ersten beiden Phasen erlangten Kenntnisse zu vertiefen, werden die Schüler nun im letzten Abschnitt mit der eigenständigen Erstellung eines Programms unter Verwendung von Fujaba life^3 konfrontiert. Das Ganze geht in Form einer Gruppenarbeit vonstatten, wobei jede Gruppe ungefähr vier Schüler umfasst und in Konkurrenz zu den anderen steht. Außerdem behandeln alle Schülergruppen das gleiche Problem. Während der Projektlaufzeit präsentiert jede der Gruppen immer wieder ihre aktuelle Lösung, die dann anschließend von der gesamten Schulklasse diskutiert wird.

Das oben genannte Werk demonstriert den konkreten Einsatz von Fujaba life^3 auch anhand eines Beispiels. Hierfür wird das Szenario eines mehrstöckigen Hauses verwendet, in dem ein Lift die sich im Haus befindlichen Personen zwischen den verschiedenen Ebenen hin- und hertransportiert.

6.5 Klassenbibliotheken

Hierbei erfolgt die Einführung der Schüler in die Objektorientierung und deren Grundprinzipien mittels entsprechender Klassenbibliotheken, bei deren Design die didaktischen Aspekte im Vordergrund standen. Im Rahmen eines vorgegebenen

Szenarios müssen die Lernenden durch Verwendung der bereitgestellten Klassen bestimmte, zunehmend schwieriger werdende Aufgaben lösen. Auf diese Art und Weise werden dabei die objektorientierten Konzepte Schritt für Schritt erarbeitet.

6.5.1 Ponto

Beim Ansatz nach [Humb06] (S. 120 ff., S. 225 f.) wird Ponto zur Erzeugung von Textdokumenten verwendet.

Es handelt sich bei Ponto um ein frei verfügbares Modul für Python, das die Kontrolle über bestimmte Objekte in OpenOffice ermöglicht. Solche Objekte lassen sich recht einfach instanzieren. Unter Verwendung der gängigen Punktnotation können dann anschließend die Methoden dieser Objekte zum Anstoßen der gewünschten Aktionen aufgerufen werden. Der Aufbau von Ponto an sich folgt streng dem objektorientierten Gedanken.

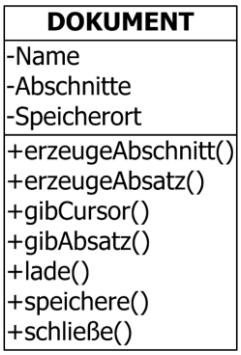


Abbildung 60: Klassendiagramm für DOKUMENT-Klasse

Ausgehend vom oben abgebildeten Klassendiagramm wird im ersten Schritt ein einfaches Textdokument erstellt. Dazu ist lediglich ein Objekt der Klasse DOKUMENT zu instanzieren, das dann auch sogleich direkt angezeigt wird. Danach werden dann durch entsprechende Methodenaufrufe die Attribute dieses Objektes verändert. Auch hierbei erhalten die Schüler eine direkte, visuelle Rückmeldung zu ihren Aktionen.

Zum besseren Verständnis folgt ein kurzes Codebeispiel für die Verwendung von Ponto, das wohl selbstklärend ist und keiner weiteren Erklärung bedarf:

```
from ponto import DOKUMENT, Zentriert

einladung = DOKUMENT()

absatz1 = einladung.erzeugeAbschnitt("Liebe Freunde!")
absatz1.setzeAusrichtung(Zentriert)
absatz1.setzeZeilenabstand(1.5)
```

Das unmittelbare Feedback nach jeder eingegebenen Zeile im Pythoninterpreter wirkt natürlich stark motivierend auf die Lernenden und erleichtert ihnen, ein Verständnis für die einzelnen Codezeilen und deren Auswirkungen zu entwickeln. Die Erarbeitung der objektorientierten Grundprinzipien erfolgt bei diesem Ansatz mit Hilfe der interaktiven Rückkoppelung.

Nachdem die Schüler ein grundlegendes Verständnis für den Aufbau und die Verwendung von Ponto sowie für die Basiskonzepte der Objektorientierung entwickelt haben, erfolgt das weitere Vorgehen ganz nach dem Prinzip des forschenden Lernens. Die Lernenden erhalten hierzu Arbeitsblätter überreicht, die sie selbstständig ausarbeiten müssen. Der Lehrer nimmt dabei lediglich die Rolle des Coachs ein und unterstützt seine Schüler durch Ratschläge zur Selbsthilfe.

6.5.2 Von Stiften und Mäusen

Dieser Ansatz stellte laut [Brin04] (S. 29 f.) das erste geschlossene Unterrichtskonzept zur Einführung von Schülern in die objektorientierte Programmierung dar. Es soll nun näher vorgestellt werden.

Eine einfach gehaltene Klassenbibliothek, die mittlerweile für viele unterschiedliche Programmiersprachen und Plattformen verfügbar ist, stellt die Grundlage dar. Sie wurde speziell nach didaktischen Gesichtspunkten gestaltet und versteckt bewusst die allzu technischen Details vor den Einsteigern.

Die Basisklassen dieser Bibliothek modellieren die Teile eines Computers, die bei der Interaktion für den Anwender von Bedeutung sind, nämlich Bildschirm, Maus und Tastatur. Zum Zwecke der Anschaulichkeit kommt noch ein Stift dazu, mit dem man auf dem Bildschirm zeichnen kann. Der davon abgeleitete Buntstift verfügt über noch mehr Funktionalitäten. Zusätzlich wird auch die abstrakte Klasse Anwendung zur Verfügung gestellt, die als Basis für weitere Anwendungen dient und über einen Bildschirm, eine Maus und eine Tastatur verfügt.

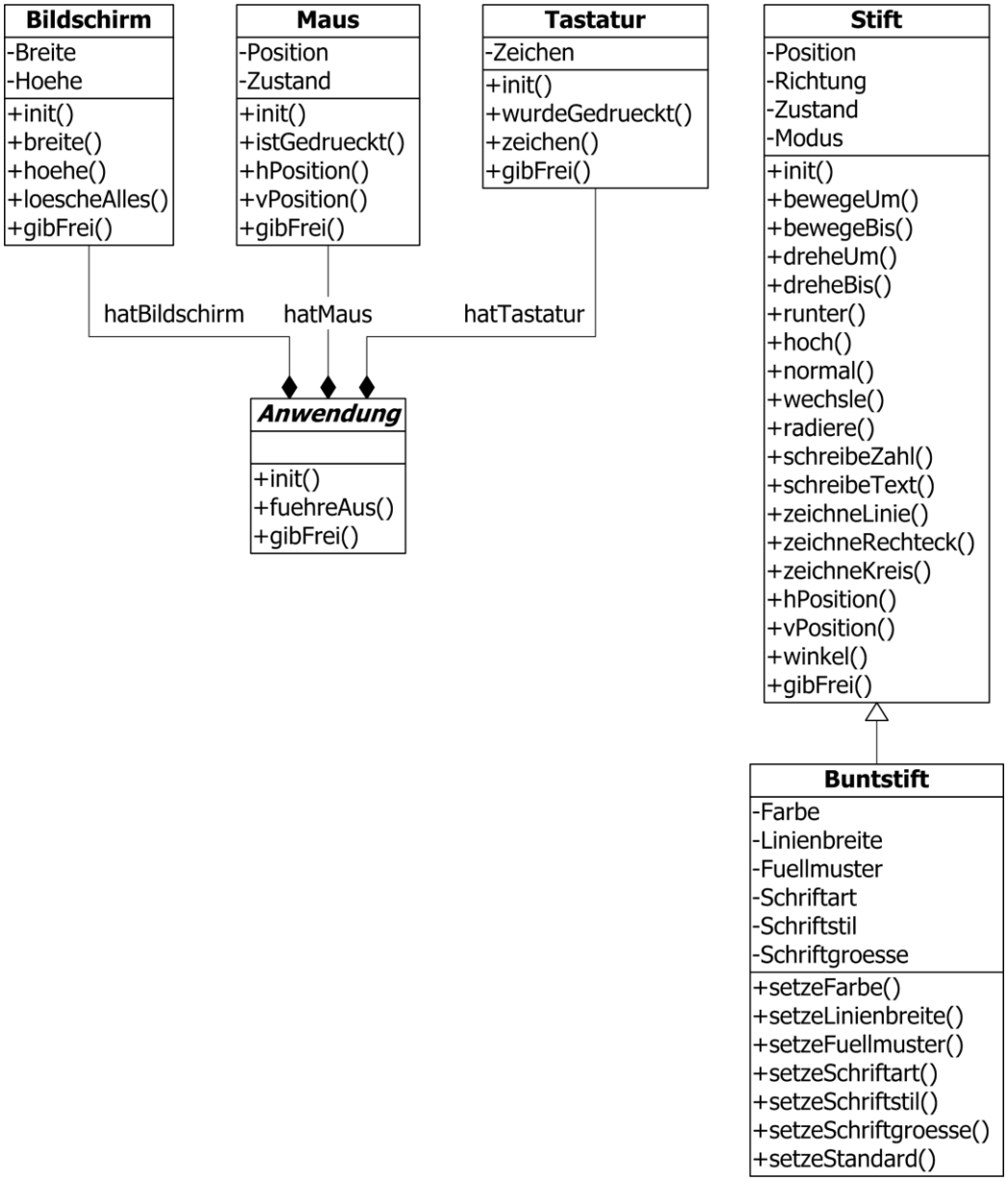


Abbildung 61: Klassendiagramm für Stifte und Mäuse-Basisklassen

Anhand dieser Basisklassen lassen sich die wichtigsten Konzepte der Objektorientierung vorstellen und demonstrieren. Von den Schülern sind unter Verwendung der beigestellten Klassenbibliothek schrittweise komplexer werdende Aufgaben auszuarbeiten. Es sind hierbei im Rahmen der Übungen verschiedene Grafiken, die sich aus Linien, Rechtecken und Kreise zusammensetzen, am virtuellen Bildschirm auszugeben. Die Komplexität der Beispiele erhöht sich dadurch, dass weitere Klassen zu den oben genannten Basisklassen hinzukommen – unter [@learc] findet sich eine Übersicht über die üblichen Stift und Mäuse-Klassen – und die Bibliothek von den Schülern auch um selbst implementierte Klassen zu erweitern ist.

Abschließend sei noch der Java-Quellcode für ein von der Klasse Anwendung abgeleitetes Programm angeführt, durch das ein Nikolaushaus gezeichnet wird:

```
public class Nikolaushaus extends Anwendung {
  public void fuehreAus() {
    stift.bewegeBis(100, 100);
    stift.runter();
```

```
    stift.bewegeBis(140, 100);
    stift.bewegeBis(100, 60);
    stift.dreheBis(60);
    stift.bewegeUm(40);
    stift.dreheUm(-120);
    stift.bewegeUm(40);
    stift.bewegeBis(100, 60);
    stift.bewegeBis(100, 100);
    stift.bewegeBis(140, 60);
    stift.bewegeBis(140, 100);
  }

  public static void main(String[] args) {
    Nikolaushaus meinNikolaushaus = new Nikolaushaus();

    meinNikolaushaus.init();
    meinNikolaushaus.fuehreAus();
  }
}
```

Die Ausführung dieser Anwendung hat die nachfolgende graphische Ausgabe am virtuellen Bildschirm zur Folge:

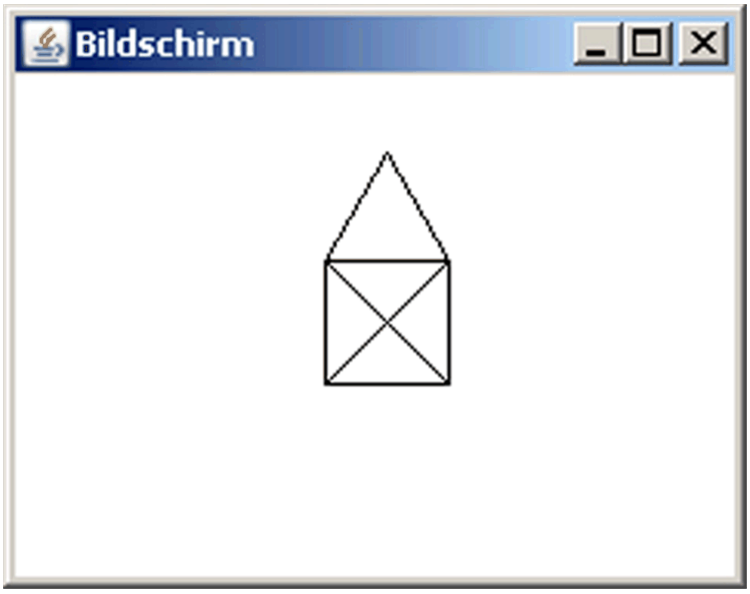


Abbildung 62: Ausgabe von Nikolaushaus am virtuellen Bildschirm

6.6 LEGO Mindstorms

Zur Vermittlung der objektorientierten Basiskonzepte wird LEGO Mindstorms eingesetzt. Die Vielzahl an zur Verfügung stehenden Programmiersprachen und -umgebungen eröffnet dem Lehrenden in Bezug auf die Unterrichtsgestaltung viele Möglichkeiten. So kann er im Anfängerunterricht mit einer graphischen Entwicklungsumgebung beginnen und dann in weiterer Folge auf eine textuelle Sprache, für die es eine entsprechende Klassenbibliothek zur Programmierung von LEGO Mindstorms gibt, umsteigen. Unabhängig davon, für welche Vorgehensweise sich der Lehrer letztendlich entscheidet, das direkte Feedback – die Schüler müssen lediglich das Verhalten der von ihnen programmierten Roboter beobachten – wirkt sich sicherlich positiv auf den Lerneffekt aus.

6.6.1 RCX / leJOS

[Rink01] (S. 44 ff.) und [Diet01] beschreiben einen Ansatz, um Schüler der zehnten Schulstufe unter Verwendung der RCX-Version von LEGO Mindstorms und der zugehörigen leJOS RCX-API in die grundlegenden Begriffe und Vorgehensweisen der Objektorientierung einzuführen. Es gelangen dabei auch zwei eigens entwickelte Programme zum Einsatz: RCXDownload erleichtert den Lernenden den Umgang mit den Java-Quelldateien und RCXDirectMode ermöglicht die direkte Steuerung des RCX-Steins.

Schwerpunktmäßig zielt das Konzept darauf ab, den Unterrichteten die Begriffe Klasse, Objekt, Vererbung und Ereignis näherzubringen. Das Ganze läuft in den folgenden sechs Phasen ab:

- Konstruktion des Roboters
- Funktionstest des Roboters
- Einführung in objektorientierte Programmierung
- Dekonstruktion und Anwendung eines Testprogramms
- Nutzung und Realisierung von Vererbung
- Einführung in Ereignisbehandlung

Im ersten Schritt stellt der Lehrer seinen Schülern das gewünschte Endprodukt vor, nämlich einen Roboter, der einer schwarzen Linie folgen kann. Weiters gibt er eine kurze Einführung in die Verwendung des Roboterbaukastens. Danach müssen die Lernenden selbstständig in Gruppen jeweils einen Roboter zusammenbauen, der sich fahrend bewegen können und über einen Tast- und einen Lichtsensor verfügen soll. Bei dieser Tätigkeit dürfen jene auch auf die Standardbaupläne zurückgreifen, da diese sowieso nur Grundgerüste beschreiben, wodurch genug Raum für eigene Kreativität bleibt.

Im Anschluss an die Konstruktionsphase unterziehen alle Gruppen ihre Roboter einem ersten Funktionstest. Das dient einerseits dazu, Konstruktionsfehler aufzeigen und andererseits, den Schülern ein rasches Erfolgserlebnis zu gönnen. Zum Testen wird dabei RCXDirectMode verwendet.

Danach müssen dann alle Schülergruppen ihre individuellen Robotermodelle auf einem Blatt Papier beschreiben und vor der restlichen Schulklasse präsentieren. Erfahrungsgemäß werden dabei die Eigenschaften und Funktionalitäten der Roboter recht zutreffend beschrieben, ohne dass hierfür viele einführende Worte vonnöten sind. Ausgehend vom Roboter als Beispiel für ein greifbares Objekt und den verfassten Beschreibungen werden nun die Begriffe des Objektes, des Attributes und des Dienstes eingeführt. Hinterher erhalten die Gruppen eine Schablone für die Roboterbeschreibung und müssen ihre ursprüngliche Darstellung dahingehend adaptieren. Von den fertig ausgefüllten Vorlagen ist es gedanklich nur mehr ein kleiner Schritt zum Klassenbegriff, der natürlich den Schülern auch in diesem Zusammenhang vorgestellt wird.

Den Schülern wird dann anschließend die in Java implementierte Klasse Roboter, welche die Dienste geh_vorwaerts() und geh_rueckwaerts() anbietet, beigestellt. Der Quelltext sieht dabei wie folgt aus:

```
import josx.platform.rcx.*;

public class Roboter {
  Motor linkerMotor = Motor.A;
  Motor rechterMotor = Motor.C;

  public void geh_vorwaerts() {
    linkerMotor.forward();
    rechterMotor.forward();
  }

  public void geh_rueckwaerts() {
    linkerMotor.backward();
    rechterMotor.backward();
  }

  public void stop() {
    linkerMotor.stop();
    rechterMotor.stop();
  }

  public void warte(int zeit) {
    try {
      Thread.sleep(zeit);
    }
    catch (Exception e) {}
    finally {
      stop();
    }
  }
}
```

Weiters wird ihnen ein Testprogramm zum Ausprobieren zur Verfügung gestellt. Die Aufgabe der Schüler besteht nun darin, den Quelltext zu kommentieren – sie erhalten dadurch einen ersten Eindruck von der Sprachsyntax – und die Implementierung mittels RCXDownload auf den Roboter hochzuladen und auszuprobieren. Dazu werden noch begleitende Übungen mit UML-Diagrammen zwecks Vertiefung der Kenntnisse über Klassen und Objekte durchgeführt.

Anhand der hinzukommenden Anforderung, dass der Roboter auch geh_links() und geh_rechts() beherrschen soll, wird das Vererbungsprinzip eingeführt. Von der Klasse Roboter wird nun die erweiterte Klasse RoboterLR abgeleitet, die ganz nach dem Vererbungsprinzip nur die neuen Methoden definiert:

```
public class RoboterLR extends Roboter {
  public void geh_links() {
    linkerMotor.backward();
    rechterMotor.forward();
  }

  public void geh_rechts() {
    linkerMotor.forward();
    rechterMotor.backward();
  }
}
```

Von den Gruppen ist das Testprogramm dahingehend zu ändern, dass ihr LEGO Mindstorms-Roboter unter Verwendung der Klasse RoboterLR ein Quadrat abfährt. Hierbei erkennen diese nun, dass die abgeleitete Klasse ebenfalls das Vorwärts- und Rückwärtsfahren ermöglicht, obwohl die zugehörigen Methoden gar nicht in dieser definiert wurden. Erfahrungsgemäß wird ihnen dadurch der Nutzen der Vererbung endgültig klar. Zur Festigung des Wissens erhalten sie noch die Aufgabe eine neue Klasse zu schreiben, die den Dienst get_quadrat() anbietet und von der Klasse RoboterLR abgeleitet ist.

Die letzte Phase beschäftigt sich mit der Einführung der Schüler in die Ereignisbehandlung. Das zugrundeliegende Prinzip wird hierbei anhand folgender Problemstellung erklärt: Ein Roboter fährt durch ein Labyrinth und muss dabei ununterbrochen – auch während er andere Befehle ausführt – den Tastsensor überprüfen und darauf reagieren. Dies kann nur mittels eines nebenläufigen Ereignismodells sichergestellt werden, das den Lernenden in Grundzügen vorgestellt wird. Sie lernen dabei am Beispiel des Tastsensors den Begriff der Ereignisquelle und anhand der Klasse TastZurueckListener den des Ereignisempfängers kennen.

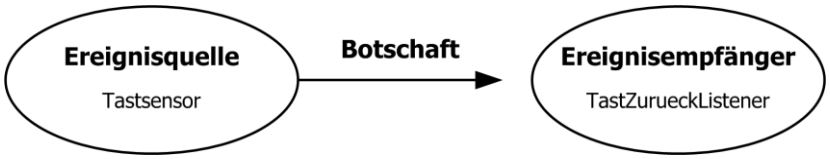


Abbildung 63: Zusammenhang zwischen Ereignisquelle und Ereignisempfänger

Zu guter Letzt werden den Schülern noch die beiden Klassen RoboterLinie und SpielfeldListener beigestellt. Erstere ist wiederum von der Klasse RoboterLR abgeleitet. Anhand dieser beiden Klassen lassen sich alle bisher gelernten Konzepte nochmals wiederholen. Zum Ausprobieren ist von den Lernenden lediglich das Testprogramm geringfügig zu adaptieren.

6.6.2 NXT / leJOS

Der Ansatz gemäß [Bagn07] (S. 58 ff.) zielt zwar vorrangig darauf ab, den Nutzern der NXT-Variante von LEGO Mindstorms einen schnellen Einstieg in die Java-Programmierung zu ermöglichen, damit jene ihre Roboter unter Verwendung der leJOS NXT-API programmieren können. Aus dieser Einführung lassen sich allerdings Teilkonzepte extrahieren, anhand derer die Vermittlung der objektorientierten Konzepte im Schulunterricht erfolgen kann. Diese sollen nun in Folge erläutert werden.

Die Motivation für die Objektorientierung wird anhand eines Roboters erklärt, der die Fähigkeit des Schachspielens besitzt. Dieser muss einerseits möglichst sinnvolle Spielzüge berechnen und andererseits die Figuren am Schachbrett bewegen können. Das objektorientierte Paradigma erlaubt nun durch die Aufteilung des Codes auf unterschiedliche Einheiten, sogenannte Objekte, eine saubere Trennung dieser Funktionalitäten voneinander. In Java wird das Verhalten solcher Objekte in Form von Klassen mit zugehörigen Attributen und Methoden definiert. Im konkreten Fall des Schachroboters lässt sich die Funktionalität auf die Hauptklassen ChessPlayer und ChessArm aufteilen, wobei erstere Klasse dabei auch auf Methoden der zweiten zurückgreift:

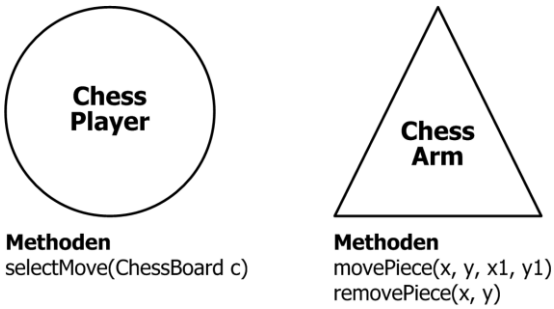


Abbildung 64: Wichtigste Klassen des Schachroboters

Im Anschluss daran wird dann das Klassenkonzept näher erläutert. Und zwar werden Klassen als spezielle Programmierstrukturen beschrieben, die das Verhalten der zugehörigen Objekte definieren und zum Erzeugen neuer Objekte mittels des Schlüsselwortes new dienen. Diese Objekte lassen sich dann für Methodenaufrufe und Variablenabfragen nutzen. Anhand der String-Klasse wird demonstriert, dass von einer Klasse beliebig viele Instanzen erzeugt werden können.

Das Vererbungsprinzip wird am Beispiel von Roboterarmen erklärt. Als Basis dient in diesem Zusammenhang ein Arm, der mittels entsprechender Motorensteuerung auf und ab bewegt werden kann und durch gleichnamige Klasse folgendermaßen definiert wird:

```
import lejos.nxt.*;

public class Arm {
  public void armUp() {
    Motor.B.forward();
  }

  public void armDown() {
    Motor.B.backward();
  }
}
```

Auf Grundlage dieses Arms soll nun ein Roboterarm entwickelt werden, der neben dem Auf- und Abbewegen auch seine Greifhand öffnen und schließen können soll. Mittels Vererbung lässt sich diese Anforderung elegant lösen. Die Klasse ClawArm wird dazu von der Klasse Arm abgeleitet und erbt dadurch all ihre Attribute und Methoden. Nun müssen lediglich die erweiterten Funktionalitäten in der neuen Klasse definiert werden, damit der erweiterte Roboterarm wie gewünscht gesteuert werden kann. In Java sieht das Ganze wie folgt aus:

```
import lejos.nxt.*;

public class ClawArm extends Arm {
  public void openClaw() {
    Motor.C.forward();
  }

  public void closeClaw() {
    Motor.C.backward();
  }
}
```

Mit Hilfe eines einfachen Klassendiagramms lässt sich der Zusammenhang folgendermaßen darstellen, wobei anzumerken ist, dass in der Programmiersprache Java die Klasse Object – nicht zu verwechseln mit dem gleichnamigen objektorientierten Konzept – automatisch als Basisklasse aller Klassen dient:

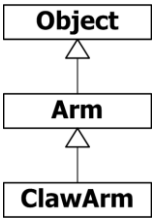


Abbildung 65: Vererbungshierarchie der Roboterarme

Abschließend werden dann noch speziellere Konzepte, wie z. B. abstrakte Klassen oder Interfaces, erklärt. Da dies aber weit über den objektorientierten Einstiegsunterricht hinausgeht, soll darauf nicht näher eingegangen werden.

6.6.3 COOL-Ansatz

Im Rahmen des Projektes Comprehensive Object-Oriented Learning (COOL) wurden diverse Experimente durchgeführt, die sich mit dem Vermitteln und Erlernen der Objektorientierung im Schulunterricht auseinandersetzten. Eines davon war der unter [Gran05] (S. 40 ff.) beschriebene Ansatz, mittels dem vierzehnjährige Schüler unter Verwendung der RCX-Variante von LEGO Mindstorms, der zugehörigen leJOS RCX-API, dem Java-Editor JCreator und RCXDownload in die objektorientierten Konzepte eingeführt wurden. Der Schwerpunkt des dreitägigen Schulversuches, der in Folge beschrieben wird, lag dabei auf der Kapselung.

Das Projekt beginnt damit, die Lernenden durch das Spielen mit den Legosteinen und dem gruppenweisen Zusammenbau eines Roboters – idealerweise bilden jeweils drei Schüler eine Gruppe – mit LEGO Mindstorms vertraut zu machen. Nach dieser Eingangsphase erhalten sie dann Beschreibungen zur Roboterprogrammierung und zur API. Diese liefern ihnen sämtliche Informationen, die sie für die weitere Arbeit benötigen. Weiters bekommt jede Schülergruppe eine Programmvorlage und ein ausführbares Testprogramm, das den Roboter für zwei Sekunden vorwärts fahren lässt. Anschließend steht den Schülern dann Zeit zum Ausprobieren zur Verfügung.

Danach müssen sie unterschiedliche Programmieraufgaben mit zunehmender Komplexität lösen. Bei der ersten Übung sollen die Lernenden lediglich die richtigen Befehle per Copy & Paste in ihre Programmvorlage übertragen. In weiterer Folge sind Sensorwerte abzufragen, Ausgaben auf das LCD zu schreiben und eigene Methoden zu definieren und implementieren, durch welche die Möglichkeiten ihres Roboters entsprechend erweitert werden. So ist dabei z. B. eine Methode zu schreiben, die den Roboter dazu veranlasst, sich um 180 Grad zu drehen. Abschließend ist der Roboter von jeder Gruppe noch derart zu programmieren, dass er einer schwarzen Linie folgen kann. Den Algorithmus dafür erarbeitet der Lehrer davor gemeinsam mit seinen Schülern.

Am dritten und letzten Tag werden dann die Roboterbaukästen beiseite gelegt und die Lernenden mit einer neuen Aufgabe konfrontiert: Und zwar müssen sie unter

Verwendung einer beigestellten Klassenbibliothek die Abläufe in einem virtuellen Restaurant nachprogrammieren. Das dient dazu, dass sie ihre Kenntnisse in Sachen Objektorientierung in einem anderen Kontext anwenden lernen und auf diese Art festigen. Zu beachten ist hierbei, dass im Rahmen der Simulation lediglich Texte am Bildschirm generiert werden.

Als Vorbereitung erhalten die Gruppen das nachfolgende Bild eines Restaurants und müssen die aus ihrer Sicht relevanten Objekte bestimmen. Die einzelnen Lösungen werden dann innerhalb der gesamten Schulklasse diskutiert.

Abbildung 66: Kristen Nygaards Restaurant der Objekte

Danach beginnt dann die eigentliche Programmierphase, wobei von den Schülern lediglich Objekte der zur Verfügung gestellten Klassen zu instanzieren und die passenden Methoden aufzurufen sind. Zur weiteren Vereinfachung erhalten alle Gruppen auch hier wieder eine Programmvorlage beigestellt.

Im Konkreten ist von den Lernenden folgende Aufgabe zu lösen: Vier Gäste betreten das Restaurant, um an einem Tisch Platz zu nehmen und bei einem Kellner eine Bestellung aufzugeben. Der Kellner gibt diese dem Küchenchef weiter, der für die virtuelle Zubereitung der Speisen sorgt. Die fertig gekochten Speisen serviert der Kellner dann den Gästen.

Die ersten Zeilen des Java-Quellcodes für eine mögliche Lösung könnten wie folgt aussehen:

```
class Control {
  Restaurant restaurant;
  Person per, lise, kjell, hanne;

  Control() {
    restaurant = new Restaurant();
    per = new Guest("Per", "male");
    lise = new Guest("Lise", "female");
    kjell = new Guest("Kjell", "male");
```

```
    hanne = new Guest("Hanne", "female");

    restaurant.butler.changeNumberOfSeats(restaurant.table4, 4);
    restaurant.butler.placeByTable(per, restaurant.table4);
    restaurant.butler.placeByTable(lise, restaurant.table4);
    restaurant.butler.placeByTable(kjell, restaurant.table4);
    restaurant.butler.placeByTable(hanne, restaurant.table4);

    // weitere Aktionen ausführen ...
  }
}
```

6.6.4 LEGO-Robotik mit Java

[@lego] stellt einen auf einer speziellen Lernumgebung basierenden Ansatz zur Einführung der Schüler in die objektorientierten Konzepte dar.

In dieser Umgebung stehen den Schülern unterschiedliche, nach didaktischen Gesichtspunkten konzipierte Java-Klassenbibliotheken zur Programmierung ihrer Roboter zur Verfügung. Die Erklärungen erfolgen auf virtuellem Wege mittels Musterbeispielen und entsprechender Kommentare auf der zugehörigen Webseite. Für die Entwicklung wird ein einfach bedienbarer Onlineeditor bereitgestellt, der verschiedene Programmvorlagen und nützliche Funktionen, wie z. B. das Kompilieren des Java-Programms, anbietet.

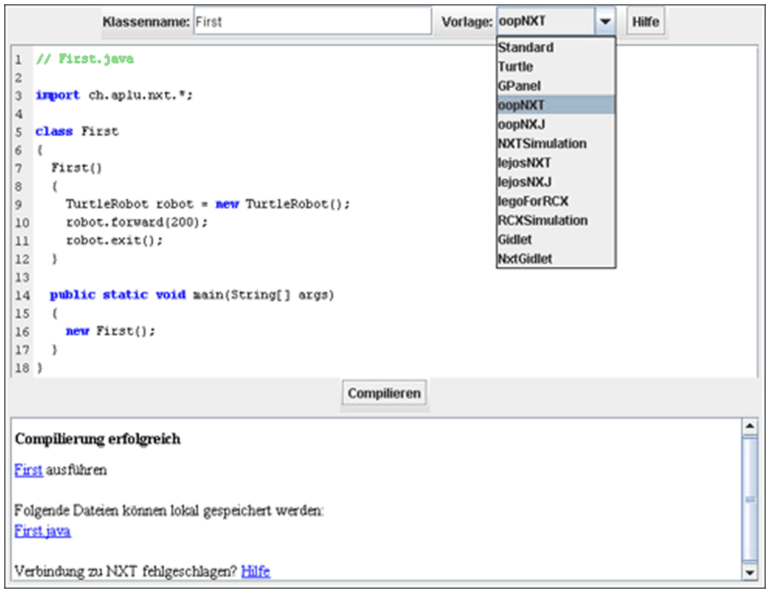


Abbildung 67: Onlineeditor beim LEGO-Robotik-Ansatz

Die Programmierung kann dabei auf drei verschiedene Arten vonstatten gehen:

- Direktor Modus: Hierbei steuert ein am Benutzerrechner laufendes Java-Programm den Roboter via Bluetooth.
- Autonomer Modus: Hierbei wird das übersetzte Java-Programm vom Benutzerrechner auf den Roboter übertragen, der es dann ausführt.
- Simulationsmodus: Hierbei werden die wichtigsten Aspekte der Robotersteuerung lediglich simuliert, wodurch kein physikalischer Roboter erforderlich ist.

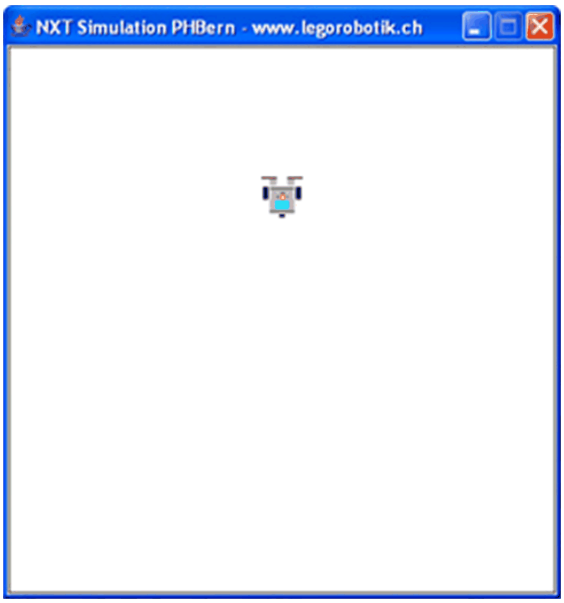


Abbildung 68: LEGO-Robotik-Simulator

Die Lernumgebung besteht aus den folgenden fünf Lernprogrammen:

- oopNXT: Es ermöglicht die Programmierung von Robotern der NXT-Version im direkten Modus unter Verwendung der Klassenbibliothek NxtJLib. Jene erweitert die Bibliothek icommand um einige nützliche Methoden und ist streng nach den objektorientierten Regeln aufgebaut.
- oopNXJ: Es ermöglicht die Programmierung von Robotern der NXT-Version im autonomen Modus unter Verwendung der Klassenbibliothek NxtJLibA. Sie stellt eine Erweiterung der Bibliothek lejosNXJ dar. Solche Programme sind praktisch identisch mit den unter Verwendung von oopNXT erstellten.
- lejosNXT: Es ermöglicht die Programmierung von Robotern der NXT-Version im direkten Modus unter Verwendung der Klassenbibliothek icommand.
- lejosNXJ: Es ermöglicht die Programmierung von Robotern der NXT-Version im autonomen Modus unter Verwendung der Klassenbibliothek lejosNXJ. Solche Programme sind nicht identisch mit den unter Verwendung von lejosNXT erstellten.
- LegoRCX: Es ermöglicht die Programmierung von Robotern der RCX-Version im direkten und im autonomen Modus.

In Folge soll nun oopNXT näher vorgestellt werden. Die anderen oben erwähnten Lernprogramme weichen nur geringfügig davon ab und werden deshalb nicht eigens behandelt.

Jeder Roboter wird durch ein Objekt der Klasse NxtRobot dargestellt. Alle Motoren und Sensoren werden ebenfalls ganz im objektorientierten Sinne in Form von Objekten repräsentiert und können mittels der Methode addPart() einem NxtRoboter-Objekt zugeordnet werden. Die Verarbeitung der von den Sensoren ausgelösten Ereignisse geschieht nach dem üblichen Java-Ereignismodell.

Die Einführung von Anfängern in die Programmierung und in die grundlegenden objektorientierten Prinzipien erfolgt anhand der Klasse TurtleRobot, welche die

Methoden forward(), backward(), left() und right() zur einfachen Steuerung des Roboters zur Verfügung stellt. Durch Musterbeispiele und dazu verfügbare Erklärungen werden den Lernenden nach und nach die Programmierkonzepte vorgestellt. Durch das Ausprobieren der Programme und Herumspielen mit den Quelltexten können die Schüler ihr Wissen vertiefen und festigen.

Als einleitendes Beispiel dient das folgende, einfache Java-Programm, das eine kurze Vorwärtsbewegung des Roboters zur Folge hat:

```
import ch.aplu.nxt.*;

class First {
  First() {
    TurtleRobot robot = new TurtleRobot();
    robot.forward(200);
    robot.exit();
  }

  public static void main(String[] args) {
    new First();
  }
}
```

Das Vererbungsprinzip wird bei diesem Ansatz anhand der Klasse AlarmRobot erklärt. Diese wird von der Klasse TurtleRobot abgeleitet und bietet zusätzlich zu dieser eine Methode alarm() an. Der Quelltext – inklusive einem Programm, das diese Klasse verwendet – sieht dazu wie folgt aus:

```
import ch.aplu.nxt.*;

public class AlarmRobotApp {
  public AlarmRobotApp() {
    AlarmRobot ar = new AlarmRobot();
    ar.forward(100);
    ar.left(90);
    ar.alarm();
    ar.backward(50);
    ar.exit();
  }

  public static void main(String[] args) {
    new AlarmRobotApp();
  }
}

class AlarmRobot extends TurtleRobot {
  public void alarm() {
    playTone(700, 1000);
  }
}
```

Anhand eines komplexeren Beispiels wird dann auch noch das Überschreiben von Methoden im Rahmen der Vererbung demonstriert. Weiters werden Musterbeispiele und Erklärungen für die Abfrage der Sensoren gegeben, die allerdings keinen Beitrag zum

besseren Verständnis der objektorientierten Basiskonzepte leisten und deshalb nicht näher betrachtet werden sollen.

6.6.5 Fujaba goes Mindstorms

Einen weiteren kreativen Ansatz für die elfte Schulstufe stellt [Diet03] vor. Hierbei wird das CASE-Tool Fujaba zum graphischen Entwurf eines Programms, das einen LEGO Mindstorms-Roboter fernsteuert, eingesetzt.

Der Technologiemix soll den Schülern einerseits die Arbeit erleichtern und sie andererseits motivieren. Fujaba ermöglicht das Erstellen von Programmen auf graphischem Wege, LEGO Mindstorms stellt eine interessante und vor allem greifbare Anwendung dar und durch das Fernsteuern wird das Auffinden von Fehlern deutlich erleichtert. Der direkte Zusammenhang zwischen Modell und Wirklichkeit lässt die Objekte für die Schüler fassbar werden, was zu einem wesentlich besseren Verständnis ihrerseits führt.

Als Szenario dient die bekannte Problemstellung der Türme von Hanoi, die von den Lernenden in Gruppen zu bearbeiten ist. Im Konkreten ist von jenen mittels Fujaba ein Programm zu entwerfen, das unter Verwendung einer beigestellten UML-LEGO-Schnittstelle einen vorgegebenen Gabelstapler derart steuert, dass dieser die besagte Aufgabe mit vier Scheiben durchspielt.

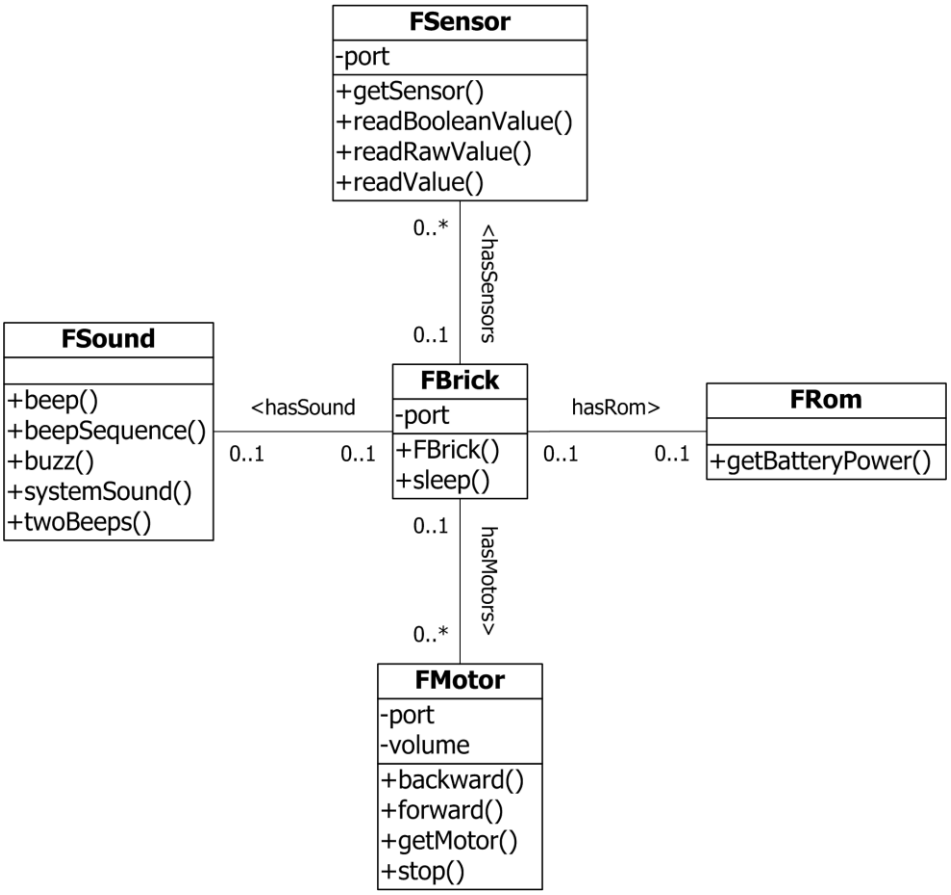


Abbildung 69: Klassendiagramm für UML-LEGO-Schnittstelle

Als Material steht jeder Schülergruppe ein Gabelstapler, eine weiße Grundplatte, drei Holzklötze als Ablagefläche, vier als Hanoi-Scheiben dienende Paletten und jede Menge färbiges Klebeband zur Wegemarkierung auf der Grundplatte zur Verfügung.

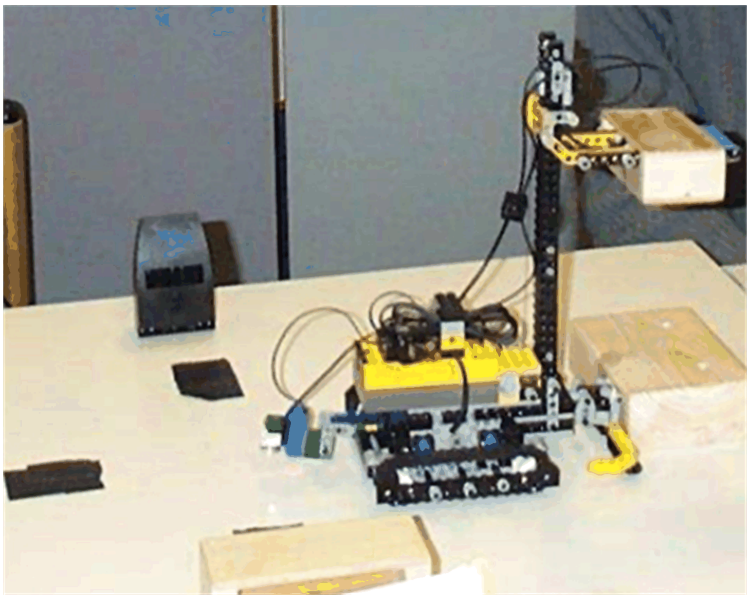


Abbildung 70: Gabelstapler bei Fujaba goes Mindstorms

Das Ganze läuft wiederum in mehreren Abschnitten ab. In der Analysephase machen sich die Schüler mit den bereitgestellten Materialen vertraut und erforschen die Problemstellung mittels eines Objektspiels. Das Gesamtproblem wird danach in einzelne Teilprobleme zerlegt und diese werden dann in textueller Form beschrieben. Am Ende dieses Abschnitts präsentieren alle Gruppen ihre Ergebnisse vor der restlichen Schulklasse.

Es folgt die Designphase, in der die textuellen Beschreibungen Schritt für Schritt mittels des CASE-Tools Fujaba in Storydiagramme umgesetzt werden. Diese lassen sich dann automatisch in Java-Code übersetzen, der mit Hilfe des dynamischen Objektbrowsers – jener ermöglicht das direkte Aufrufen einzelner Methoden – getestet werden kann. Auch hier demonstriert zum Abschluss jede der Schülergruppen ihre Resultate.

Am Ende treten dann die einzelnen Projektgruppen im Rahmen eines Turniers gegeneinander an. Sieger wird dabei jene Gruppe, deren Gabelstapler am schnellsten das Problem der Türme von Hanoi löst, wobei manuelle Eingriffe zwecks Fehlerkorrektur mit Zeitstrafen versehen werden. Vorher haben alle Gruppen noch Zeit für ausführliche Tests sowie zur Optimierung ihrer erarbeiteten Programmlösungen. Erfahrungsgemäß wird dieser Wettkampf von allen Teilnehmern recht ehrgeizig verfolgt.

Zur Nachbereitung müssen die Schüler noch das zugehörige Phasendokument vervollständigen und eine Präsentation und ein Poster über ihre Arbeit erstellen.

7 Analyse von LEGO Mindstorms zur Vermittlung objektorientierter Konzepte

Dieses Kapitel untersucht, inwieweit sich LEGO Mindstorms für die Vermittlung der Grundprinzipien der Objektorientierung eignet. Einleitend wird dazu der Einsatz der Roboterbaukästen im Schulunterricht allgemein betrachtet. Danach wird für jedes der Basiskonzepte analysiert, ob es mit Hilfe von LEGO Mindstorms den Schülern sinnvoll nähergebracht werden kann. Anschließend werden verschiedene Sprachen und Umgebungen zur Roboterprogrammierung kritisch beleuchtet, ehe die im vorherigen Abschnitt vorgestellten konkreten Ansätze – und zwar nur jene, bei denen LEGO Mindstorms zum Einsatz gelangt – genauer beleuchtet werden.

7.1 LEGO Mindstorms allgemein

Die Verwendung von LEGO Mindstorms als Lernbehelf im Unterricht ist unanhängig vom zu vermittelnden Stoff mit diversen Pro und Kontras verbunden. Diese sollen nun behandelt werden und treten natürlich auch bei der Nutzung der Roboterbaukästen im objektorientierten Anfangsunterricht zu Tage.

Gemäß [Diet01] wirken sich Roboterbaukästen stark motivierend auf die Schüler aus und ihr Einsatz im Schulunterricht ist deshalb als sinnvoll zu betrachten.

[Barn02] sieht im direkten Feedback, das die Roboter den Schüler zurückliefern, ein großes Plus. Um festzustellen, ob ihr Programm entsprechend funktioniert, müssen sie lediglich das Verhalten ihres LEGO Mindstorms-Roboters beobachten. Weiters übt die Kontrolle über ein physikalisch angreifbares Modell eine gewisse Faszination auf die Lernenden aus, weshalb sie Roboter entsprechenden Simulationen am Bildschirm vorziehen. Ebenfalls ist es als positiv anzusehen, dass bei solchen Baukästen weder Lehrer noch Schüler über die hardwaremäßige Umsetzung Bescheid wissen müssen und ohne großes Vorwissen ausgeklügelte Modelle konstruieren und programmieren können. Die vielfältigen Möglichkeiten zur Programmierung werden ebenfalls erwähnt.

Laut [Lawh02] ist das Testen von großer Bedeutung für den Entwicklungsprozess und dabei kann beobachtet werden, dass dies in Zusammenhang mit LEGO Mindstorms den Schülern Spaß bereitet. Zusätzlich lernen sie dabei auch von Anfang an auf ein robustes Programmdesign zu achten, um z. B. das Herunterfallen ihres Robotermodells vom Tisch zu vermeiden.

Durch den Zusammenbau und die Programmierung der Roboter lernen die Unterrichteten gemäß [Hirs03] wichtige Ingenieursprinzipien, wie z. B. das Planen oder das Arbeiten in Teams, kennen. Des Weiteren ist der Schwierigkeitsgrad einer Aufgabe aufgrund der vielfältigen Verwendungsmöglichkeiten von LEGO Mindstorms ganz dem Alter der Schüler anpassbar. Außerdem kann durch Verwendung der Baukästen eine Brücke zur Überwindung der Kluft zwischen Konzept und Praxis geschlagen werden.

[Kure06] beschäftigt sich mit dem allgemeinen Einsatz von Robotern im Schulunterricht und sieht es als positiv an, dass die Schüler bei deren Programmierung sowohl Software- als auch Hardwarebelange kennenlernen. Durch den Programmupload wird auch erkennbar, dass Software und Hardware voneinander unabhängig sind. Weiters

hat der Ansatz des Versuchs und Irrtums, der bei der Roboterprogrammierung seitens der Schüler großteils zum Einsatz gelangt, ein besseres Verständnis zur Folge.

Gemäß [Mcna06] (Frank Klassner) ist der Trend zu beobachten, dass neben dem klassischen Desktop diverse andere Ausprägungen des Computers, wie z. B. mobile Geräte, eine immer wichtigere Rolle einnehmen. LEGO Mindstorms eignet sich dazu, den Schülern neben dem Desktop eine weitere interessante Anwendung des Computers näherzubringen. Vorteilhaft ist dabei, dass die Plattform relativ günstig, einfach zu programmieren, vielseitig einsetzbar, offen, skalierbar und auch für bereits erfahrene Schüler noch herausfordernd ist.

Zusätzlich zu den bisher genannten Pluspunkten lässt sich noch ergänzen, dass sich LEGO Mindstorms ideal für im Bereich des entdeckenden Lernens angesiedelte Projekte eignet und durch eine geschickte Aufgabenstellung die Gestaltung eines abwechslungsreichen Unterrichts ermöglicht.

Den vielen positiven Stimmen stehen aber auch kritische Meinungen gegenüber, die dem Leser nicht vorenthalten werden sollen.

So werden von [Brin04] (S. 31) die hohen Anschaffungskosten als problematisch angesehen. Außerdem ist das Zusammenhalten der Materialen mehrerer Baukästen mit entsprechendem Aufwand verbunden.

Laut [Gran05] (S. 2) macht die Verwendung von LEGO Mindstorms im Unterricht Kindern im Alter von elf bis fünfzehn Jahren zwar anfangs Spaß, nach einigen Projekttagen ist allerdings ein deutlicher Motivationsverlust spürbar. Mädchen sind im Vergleich zu den gleichaltrigen Burschen von Haus aus weniger für die Roboterbaukästen zu begeistern.

Die hohen Kosten für den Ankauf der Baukästen verglichen mit ähnlichen Programmvisualisierungstools nennt auch [Mcna06] (Michael Goldweber) als großen Nachteil. So ist es nicht möglich, alle Schüler mit einem solchen Roboterbaukasten auszurüsten, wodurch deren Verwendung an die Öffnungszeiten des Computerlabors gebunden ist. Dies hat aber auch zur Folge, dass sich bestimmte Experimente, wie z. B. Projekte außerhalb der Schulzeit, nicht bzw. nur mit Umständen damit durchführen lassen.

Gemäß [Mcna06] (Barry Fagin) ist es nicht erwiesen, dass LEGO Mindstorms zu einem besseren Lerneffekt führt. Des Weiteren ist der Prozess des Testens und der Programmanpassung mit mehr Aufwand verbunden, als dies bei einem reinen Softwaresystem der Fall wäre. Nach jeder Änderung muss das adaptierte Programm auf den Roboter übertragen werden – außer man verwendet eine Programmierumgebung, die das Fernsteuern des Roboters ermöglicht –, was sich bereits nach kurzer Zeit als nervend entpuppt.

Als weiteren negativen Punkt lässt sich die Ungenauigkeit der in den Baukästen enthaltenen Sensoren nennen, die nach verflogener Anfangseuphorie einigermaßen frustrierend ist. So wird bei erfahrenen Benutzern der Großteil der Zeit nicht für die eigentliche Programmierung, sondern für das Herausfinden der richtigen Grenzwerte aufgewendet.

7.2 Objektorientierte Konzepte

Im Folgenden wird für jedes der Basiskonzepte – unabhängig von der zur Programmierung eingesetzten Sprache und Umgebung – untersucht, inwieweit LEGO Mindstorms für dessen Vermittlung einen positiven Beitrag leisten kann. Aufgrund der engen Verdrahtung zwischen den Begriffen Objekt und Klasse werden diese beiden Prinzipien der Objektorientierung gemeinsam betrachtet.

7.2.1 Objekte und Klassen

Gemäß [Diet01] und [Barn02] tragen das Vorhandensein der realen Roboterobjekte und die Möglichkeit, diese bei Bedarf sogar anfassen zu können, deutlich zum Verständnis dafür bei, was man sich unter dem Begriff eines Objektes vorzustellen hat. Weiters wird den Unterrichteten dadurch auch die für die objektorientierte Programmierung notwendige Abstraktion erleichtert.

Laut [Lawh02] lässt sich für die Schüler anhand der Robotermodelle leicht nachvollziehen, dass ein Objekt einen Status kapselt und das Verhalten eines jeden Objektes von seinem aktuellen Zustand abhängig ist. Außerdem demonstrieren die realen Roboterobjekte den Anwendern die wichtigsten Konzepte der Objektorientierung.

Aus pädagogischer Sicht ist es gemäß [Mcna06] (Michael Goldweber) schlecht, dass der Roboter ein Singleton darstellt, wenngleich er die physische Instanz eines Objektes repräsentiert und durch eine Hierarchie von Klassen programmiert wird. Das vermittelt den Schülern nämlich möglicherweise ein falsches Bild von Objekten.

Ohne Zweifel lässt sich das Wesen eines Objektes – es kapselt Daten und Methoden – mit Hilfe von LEGO Mindstorms sehr gut demonstrieren. Schrittweise kann man sich dabei durch entsprechende Modellierung der realen Roboterobjekte den virtuellen, informatischen Kongruenzen annähern.

Darüber hinaus lässt sich anhand der LEGO-Metapher auch der modulare Aufbau eines Programms (vgl. Roboter) auf Basis von Objekten (vgl. Roboterobjekte) sehr gut zeigen und erklären. Dass die Schüler dafür ein Verständnis entwickeln, ist für das objektorientierte Denken sehr bedeutend.

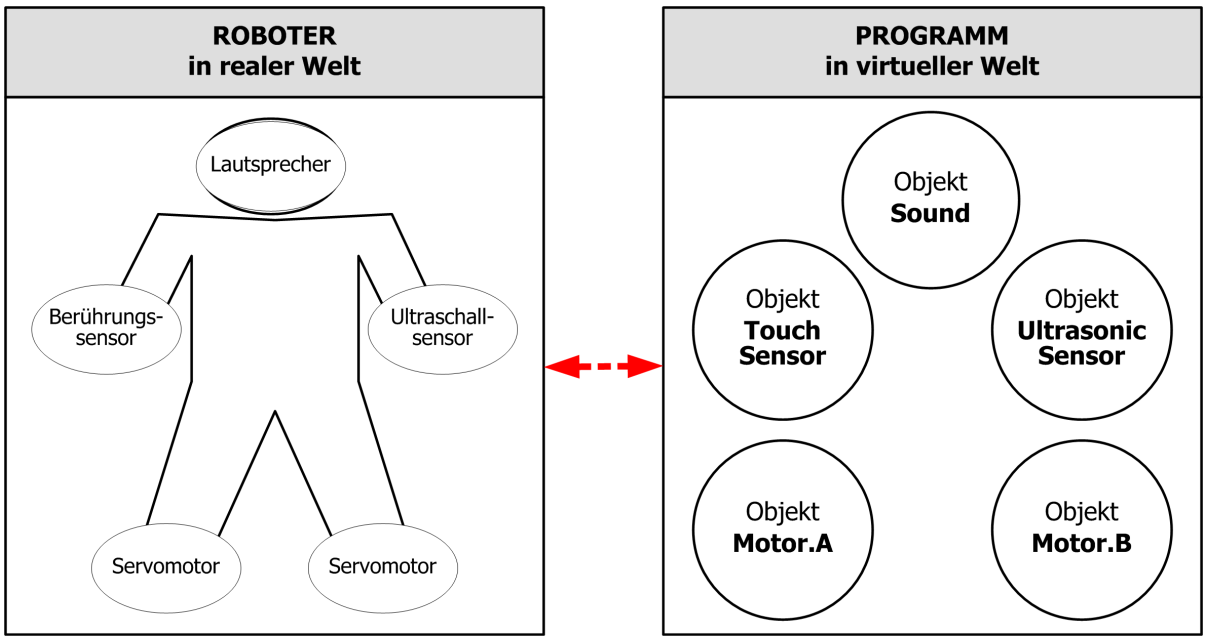


Abbildung 71: Zusammenhang zwischen Roboterobjekten und virtuellen Objekten

Hat man mit den Unterrichteten den Objektbegriff erarbeitet, dann kann diesen folgend der Klassenbegriff nähergebracht werden. Hierzu empfiehlt es sich von den einzelnen, realen Roboterobjekten ausgehend deren Gemeinsamkeiten zu betrachten. So werden die Schüler z. B. anhand der drei baugleichen Servomotoren feststellen, dass sie zwar unterschiedliche Objekte darstellen, aber ihnen doch sehr viel gemeinsam ist und sie demnach der gleichen Klasse angehören.

Fraglich ist in diesem Zusammenhang aber sowieso, ob dem begrifflichen Unterschied zwischen Objekt und Klasse überhaupt so viel Bedeutung zugemessen werden sollte. Manche Programmiersprachen, wie z. B. LPC, unterscheiden diese Begriffe nämlich bis heute nicht.

7.2.2 Polymorphismus

In Bezug auf den Polymorphismus kann LEGO Mindstorms per se nichts zur Vermittlung beitragen. Jenes objektorientierte Prinzip lässt sich allerdings mittels entsprechender Beispiele auf Sprachebene demonstrieren, wobei die Wahl einer passenden Programmiersprache dafür Voraussetzung ist.

Wesentliche Grundlage für die Beschäftigung mit diesem Konzept ist, dass die Schüler das Vererbungskonzept verstanden haben. Ist dies der Fall, dann lassen sich bestimmte Arten des Polymorphismus, wie z. B. das Überladen, recht einfach auf sprachlicher Ebene demonstrieren und sollten auch für die Schüler verständlich sein.

7.2.3 Vererbung

Das Prinzip der Vererbung lässt sich nach [Lawh02] mittels LEGO Mindstorms sehr gut demonstrieren. Es wird in diesem Zusammenhang vorgeschlagen, einen Roboter zu erstellen und diesem dann eine weitere Funktionalität hinzuzufügen. Diese physikalische Erweiterung lässt sich programmmäßig durch Ableitung des neuen, erweiterten Roboterprogramms vom bereits vorhandenen entsprechend abbilden und dadurch den Schülern demonstrieren.

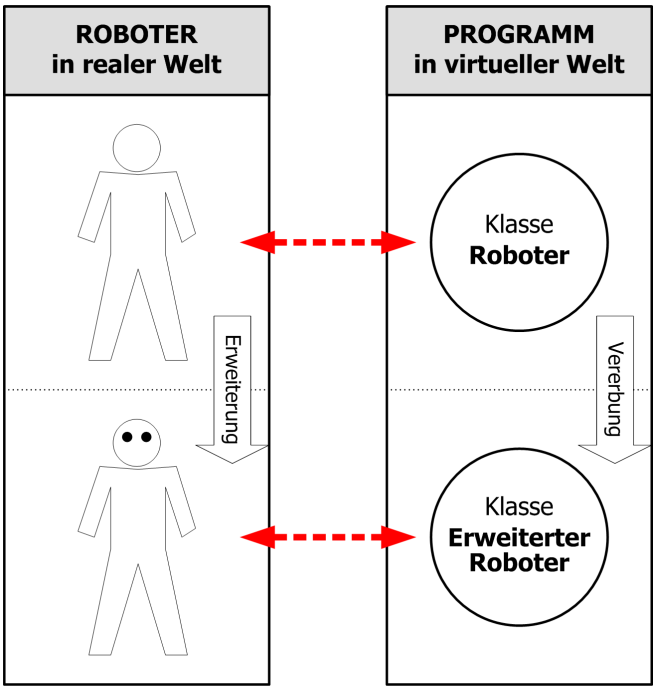


Abbildung 72: Zusammenhang zwischen Robotererweiterung und Vererbung

Diese Vorgehensweise scheint vernünftig zu sein. Auf alle Fälle ist dabei aber darauf zu achten, dass die zur Entwicklung eingesetzte Programmiersprache und -umgebung ein entsprechendes Vererbungskonstrukt anbietet und das Ganze nicht mittels Copy & Paste und anschließender Programmanpassung gelöst wird. Gemäß [Perr98] ist zweitere Vorgehensweise nämlich weit verbreitet, obwohl sie eigentlich tunlichst vermieden werden sollte.

7.3 Programmiersprachen und -umgebungen

Für die Programmierung der LEGO Mindstorms-Roboter stehen unterschiedliche Sprachen und Umgebungen zur Verfügung. Inwieweit sich die beiden Vertreter NXT-G und leJOS NXJ für den objektorientierten Anfangsunterricht eignen, wird im Folgenden untersucht.

7.3.1 NXT-G

Im ersten Schritt wird diese graphische Programmieroberfläche nun gemäß der in Kapitel 5.4.1 eingeführten Kriterien zwecks Orientierung bewertet:

	Erfüllung	Anmerkung
Komplexität reduzieren	✓	Die Komplexität wird dadurch reduziert bzw. versteckt, indem die hardwaremäßige Umsetzung vor dem Benutzer vollkommen verborgen wird.
Komplexität verstecken	✓	
Visualisierung	✓	Es wird nicht nur das vom Benutzer erstellte Programm auf visuelle Art und Weise angezeigt, sondern auch der Programmablauf kann durch Beobachtung des Roboterverhaltens visuell nachvollzogen werden.
Kleiner Sprachumfang	✓	Der verfügbare Sprachumfang – in Zusammenhang mit einer graphischen Programmierumgebung wäre es wohl sinnvoller

		von Funktionalitätsumfang zu sprechen – ist auch für Anfänger überschaubar.
Einfache Programmierumgebung	✓	Die Programmierumgebung ist einfach, übersichtlich und benutzerfreundlich gestaltet.
Benutzerfreundlichkeit	✓	
Integrierte Werkzeuge	✓	Es werden dem Benutzer alle für die Programmierung notwendigen Werkzeuge, wie z. B. die Möglichkeit zum Programmupload, geboten.
Unterstützung von Objekten	✓	Jedes reale Roboterobjekt wird in der Programmierumgebung durch einen entsprechenden Block gekapselt, den man als visuelles Objekt auffassen kann.
Unterstützung bei Codewiederverwendung	✗	Für die Codewiederverwendung gibt es programmseitig keinerlei Unterstützung.
Unterstützung beim Erlernen	✓	Es stehen dem Benutzer mehrere Einführungshilfen zur Verfügung, die ihm einen raschen Einstieg ermöglichen sollten.
Unterstützung bei Gruppenarbeit	✗	Für die Gruppenarbeit gibt es programmseitig keinerlei Unterstützung.
Verfügbarkeit	✗	Die Programmierumgebung wird gemeinsam mit dem Baukasten ausgeliefert und ist nicht frei verfügbar.

Tabelle 9: Bewertung der Anfängertauglichkeit von NXT-G

In Verbindung mit Robolab, das ebenfalls eine graphische Programmierumgebung für LEGO Mindstorms darstellt, meint [Gran05] (S. 93), dass es sich zur Vermittlung der objektorientierten Konzepte nicht eignet. Die Pilot-Version ist zu einfach und nicht flexibel genug und die Inventor-Version enthält zu viele Bugs.

Abgesehen von den Fehlern treffen die genannten Punkte auch auf NXT-G zu. Mit viel Bemühen kann man den Schülern damit noch das Konzept des Objektes und der Klasse näherbringen. Spätestens beim Versuch, ihnen damit auch den Polymorphismus oder die Vererbung zu erklären, ist ein Scheitern allerdings vorprogrammiert.

NXT-G eignet sich aber natürlich sehr wohl zur allgemeinen Einführung der Schüler in den Algorithmenbegriff sowie in die wichtigsten Programmierkonzepte, wie z. B. Kontrollstrukturen. Zur Vermittlung der grundlegenden Prinzipien der Objektorientierung muss man sich aber nach einer anderen Programmiersprache und -umgebung umschauen.

7.3.2 leJOS NXJ

Auch leJOS NXJ wird einleitend gemäß der Kriterien des Kapitels 5.4.1 beurteilt, wobei die Bewertung bestimmter Merkmale für die zugrundeliegende Programmiersprache Java geschieht:

	Erfüllung	Anmerkung
Komplexität reduzieren	✓	Auch hier wird die hardwaremäßige Umsetzung vor dem Benutzer verborgen, was eine Reduktion bzw. ein Verstecken der Komplexität zur Folge hat.
Komplexität verstecken	✓	

Visualisierung	✓	Der Programmcode an sich lässt sich hier nur unter Verwendung einer entsprechenden Java-Programmierumgebung visuell darstellen. Der Ablauf des Programms kann natürlich wieder durch Beobachtung des Roboterverhaltens visuell nachvollzogen werden.
Kleiner Sprachumfang	✓	Im Vergleich zur Programmiersprache Java ist der Sprachumfang von leJOS NXJ deutlich reduziert und einigermaßen überschaubar.
Einfache Programmierumgebung	✗	Für die Programmierung an sich kann natürlich jede beliebige Java-Entwicklungsumgebung genutzt werden. Aufgrund des verfügbaren leJOS NXJ-Eclipse-Plugins empfiehlt sich die Verwendung von Eclipse, die jedoch für Anfänger nicht unbedingt einfach zu nutzen ist. Wird eine andere Umgebung als Eclipse verwendet, dann muss man sich nach zusätzlichen Tools für den einfachen Programmupload umschauen. Eine Entwicklungsumgebung, die sowohl alle benötigten Funktionen zur Roboterprogrammierung abdeckt, als auch einfach ist, ist derzeit aber nicht verfügbar.
Klare Konzepte	✓	Die Programmiersprache Java erfüllt im Großen und Ganzen all diese Kriterien.
Reine Objektorientierung	✓	
Sicherheit bzgl. Fehler	✓	
Hohes Level bzgl. Maschinenabstraktion	✓	
Einfaches Objekt- bzw. Ausführungsmodell	✓	
Lesbare Syntax	✗	Speziell für Anfänger ist die Syntax von Java-Programmen nicht unbedingt einfach zu lesen.
Keine Redundanz	✗	Viele Probleme lassen sich unter Verwendung von Java auf unterschiedliche Arten lösen. Ob dies allerdings für eine Anfängersprache wirklich nachteilig ist, kann hinterfragt werden, denn großteils werden sich die Schüler sowieso einen vorgestellten bzw. selbst erarbeiteten Lösungsweg quasi als Schablone einprägen.
Einfacher Wechsel zu anderen Sprachen	✗	Die Umsetzung der grundlegenden Programmierkonstrukte und der objektorientierten Konzepte ist jener in vergleichbaren Sprachen sehr ähnlich, weshalb der Umstieg für erfahrende Benutzer mit nicht allzu großen Problemen behaftet sein sollte. Es ist aber eher nicht davon auszugehen, dass dies auch für Anfänger gilt.
Unterstützung bei Korrektheitszusicherung	✓	Mittlerweile wird dieses Merkmal auch von Java unterstützt. Ob man dieses Thema im Anfangsunterricht bringen muss, ist allerdings zu hinterfragen.

Tabelle 10: Bewertung der Anfängertauglichkeit von leJOS NXJ

Gemäß [Rink01] (S. 8 ff.) ist leJOS RCX – die Vorgängerversion von leJOS NXT – sehr gut dazu geeignet, um Schüler in Java und die damit verbundene objektorientierte Programmierung einzuführen. Die Roboter und deren zugehörige Komponenten ermöglichen einen einfachen Zugang zu den Sprachkonzepten des Objektes, der Klasse, der Nachricht und der Vererbung. Unter Verwendung der leJOS RCX-API kann das Eintauchen in die objektorientierte Welt außerdem beliebig tief erfolgen.

Auch [Gran05] (S. 93) spricht sich für eine Einführung der Lernenden in die objektorientierten Konzepte mittels leJOS NXJ aus, wenngleich das Erlernen von Java für Anfänger mit Problemen verbunden ist. Vor allem lernen sie jene dadurch das Prinzip der Kapselung in Zusammenhang mit Objekten kennen.

Mittels entsprechender, schülergerechter Beispiele können den Schülern unter Verwendung der leJOS NXJ-API sämtliche objektorientierte Basiskonzepte nähergebracht werden. Zu beachten ist dabei aber natürlich, dass man sich plötzlich in der Java-Welt wiederfindet – und zwar mit all seinen Vor- und Nachteilen. Für den Anfängerunterricht ist auf alle Fälle die Verwendung einer schülergerechten Java-Entwicklungsumgebung wichtig.

7.4 Konkrete Ansätze

Die im Kapitel 6.6 vorgestellten konkreten Ansätze, im Rahmen derer LEGO Mindstorms als Lernbehelf eingesetzt wird, werden in Folge einer genaueren Betrachtung unterzogen.

7.4.1 RCX / leJOS

Der grundsätzliche Aufbau dieses Ansatzes kann auf alle Fälle als sehr gelungen bezeichnet werden. Einerseits werden den Schülern klare Zwischenziele zur Orientierung vorgegeben, andererseits können sie dank der konstruktivistischen Vorgehensweise den Weg dorthin selbst bestimmen und haben genug Freiraum zum selbstständigen, kreativen Entdecken. Des Weiteren werden durch die Form Gruppenarbeit neben den technischen auch die sozialen Fähigkeiten trainiert und verbessert.

Auch die Annäherung an die Programmiersprache Java ist durch die beigestellten, ausführbaren Programme gut gelöst. Durch die anfänglich verlangte Kommentierung der Quelltexte und die Durchführung kleiner Änderungen werden die Unterrichteten bei diesem Ansatz langsam an die Java-Syntax herangeführt.

In Zusammenhang mit der Vermittlung der objektorientierten Basiskonzepte fällt positiv auf, dass den Schülern mehrere Zugänge zum Erlernen und Verstehen des Objektbegriffs geboten werden. Auch das Klassenkonzept wird gemeinsam erarbeitet und den Lernenden nicht einfach nur vorgesetzt. Die zur Vermittlung der Vererbung gewählte Vorgehensweise scheint ebenfalls schlüssig zu sein und findet sich in dieser bzw. ähnlicher Art und Weise sehr oft in der Literatur. Nicht eingegangen wird allerdings auf den Polymorphismus, stattdessen werden die Schüler in die Ereignisbehandlung eingeführt. Diese stellt zwar ein wichtiges Programmierkonzept dar, zählt jedoch nicht zu den objektorientierten Kernprinzipien.

Resümierend kann die gewählte Vorgehensweise als sehr gelungen bezeichnet werden, wenngleich sie gemäß [Diet01] noch ausbaufähig ist.

7.4.2 NXT / leJOS

Dieser Ansatz zielt darauf ab, die Benutzer des LEGO Mindstorms-Baukastens in die Java-Programmierung einzuführen. Es darf deshalb nicht verwundern, dass hier

formaler vorgegangen wird als beim vorherigen, speziell für den Schulunterricht konzipierten Modell und dass man lediglich Teilaspekte daraus unverändert im objektorientierten Einstiegsunterricht bringen kann.

Als vorrangige Motivation für die Beschäftigung mit der Objektorientierung wird genannt, dass man dank der Objekte den Quellcode sehr gut in überschaubare Einheiten aufteilen kann. Hier ist auf jeden Fall kritisch zu hinterfragen, ob dadurch nicht ein falsches bzw. zu vereinfachtes Bild von Objekten und Klassen gezeichnet wird, denn das objektorientierte Konzept zielt auf mehr als nur auf die Modularisierung ab. Die Einführung des Objekt- und Klassenbegriffes erfolgt anhand des Beispiels eines Roboters, der Schach spielen kann. Für einen Einsteiger in die Objektorientierung werden die spärlichen Erklärungen für ein Verständnis allerdings kaum ausreichen. Sinnvoller wäre es sicherlich gewesen, die realen Roboterobjekte heranzuziehen und anhand deren diese beiden Prinzipien zu erläutern. Sehr anschaulich ist hingegen die Vererbung erklärt. Das Wesen des Polymorphismus lernt man dann gegen Ende des Kapitels kennen.

Zusammenfassend lässt sich festhalten, dass die formale Einführung in das Objekt- und Klassenkonzept für den Schulunterricht nicht geeignet ist. Das Vererbungsprinzip lässt sich durch die gewählte Vorgehensweise aber sicherlich verständlich übermitteln.

7.4.3 COOL-Ansatz

Auch hier kann die grundsätzliche Vorgehensweise als gelungen bezeichnet werden. Der Ansatz lässt den Schülern genügend Platz, um die objektorientierte Welt selbst forschend entdecken zu können, und durch die Gruppenarbeit werden wiederum ihre sozialen Fertigkeiten trainiert. Weiters werden ihnen klare Zwischenziele vorgegeben und die Aufgaben sind so gestaltet, dass sich die Komplexität laufend steigert.

Durch das Beistellen einer ausführlicher Dokumentation, einer Vorlage zur Programmerstellung und einem Testprogramm wird den Unterrichteten der Einstieg in die Java-Programmierung deutlich vereinfacht. Die Wahl von JCreater als Entwicklungsumgebung und RCXDownload zur einfachen Handhabung des Programmuploads scheint ebenfalls vernünftig.

Interessant, aber auch mit Risken verbunden, ist bei diesem Ansatz der Kontextwechsel nach zwei Projekttagen von LEGO Mindstorms zum virtuellen Restaurant. Dadurch lernen die Schüler die objektorientierte Programmierung auch anhand abstrakter, nicht angreifbarer Objekte kennen und können ihr bis dahin in Sachen Objektorientierung erlangtes Wissen nochmals unter einem anderen Gesichtspunkt anwenden und vertiefen.

Aus didaktischer Sicht ist wiederum positiv anzumerken, dass den Schülern unterschiedliche Zugänge zum Objekt- und Klassenkonzept geboten werden. Der Schwerpunkt liegt bei diesem Ansatz aber eindeutig auf der Kapselung. Das Prinzip der Vererbung wird nur am Rande behandelt, der Polymorphismus gänzlich vernachlässigt.

Laut [Gran05] (S. 93 ff.) verstanden die Schüler während des durchgeführten Experiments die Programmstruktur nur zum Teil und es war zu beobachten, dass sich diese Probleme noch vergrößerten, sobald abstrakte Vorgänge abzubilden waren.

Anhand der geschilderten Probleme lässt sich erkennen, dass den Schülern nicht die objektorientierten Konzepte an sich, sondern vor allem deren konkrete Umsetzung in Java Probleme bereitete.

7.4.4 LEGO-Robotik mit Java

Dieser Ansatz unterscheidet sich grundsätzlich von den anderen, da er in Form eines Onlinekurses organisiert ist. Dieser zielt wiederum darauf ab, den Schülern die Programmierung von LEGO Mindstorms mittels Java näherzubringen.

Die Umsetzung des virtuellen Lehrgangs ist im Grunde gut gelungen. Der Onlineeditor stellt eine einfache, aber ausreichende Programmierumgebung bereit, dessen Einsatz im Unterricht mit wenig Wartungsaufwand verbunden ist. Die drei unterschiedlichen Modi erlauben bei der Unterrichtsgestaltung entsprechende Flexibilität und die Klassenbibliothek ist nach didaktischen Gesichtspunkten aufgebaut. Des Weiteren werden viele Programmbeispiele mit zugehörigen Erklärungen und Aufgaben mit steigendem Schwierigkeitsgrad geboten.

Es wäre jedoch eventuell sinnvoll gewesen, den Onlineeditor dahingehend zu erweitern, dass der Programmstart ohne einer main-Methode interaktiv erfolgen kann. Natürlich hätte man sich dadurch aber vom syntaxmäßig korrekten Java entfernt.

In Zusammenhang mit der Vermittlung der objektorientierten Konzepte fällt negativ auf, dass auf das Wesen des Objektes und der Klasse nicht näher eingegangen wird, was allerdings für das Schülerverständnis von großer Bedeutung wäre. Der Sinn der Vererbung und das Überschreiben von Methoden – eine Form des Polymorphismus – werden hingegen anschaulich demonstriert.

Als Resümee lässt sich sagen, dass das zur Verfügung gestellte Material für eine schülergerechte Einführung in die objektorientierten Konzepte nicht ausreicht. Sehr wohl aber weist der virtuelle Lehrgang genügend Potential auf, so dass darauf aufbauend ein entsprechender Ansatz für den objektorientierten Anfängerunterricht entwickelt werden kann.

7.4.5 Fujaba goes Mindstorms

Die gewählte Vorgehensweise mit Gruppenarbeit, selbstständigem Erarbeiten der Lösungen, gemeinsamem Feedback seitens der Klasse und abschließendem Turnier ist wiederum als gelungen anzusehen.

Als Entwicklungswerkzeug gelangt Fujaba zum Einsatz. Durch dessen Verwendung verlagert sich der Schwerpunkt von der objektorientierten Codierung hin zur objektorientierten Modellierung, was für die Schüler eine deutliche Vereinfachung bedeuten sollte. Um zur Lösung der Aufgabe zu gelangen, müssen sie die verfügbaren Objekte lediglich mehr in sinnvoller Art und Weise miteinander interagieren lassen. Der direkte Zusammenhang zwischen den realen Roboterobjekten und den zugehörigen Modellen am Bildschirm trägt dabei nicht nur zu einem besseren Verständnis seitens der Lernenden bei, sondern hat gemäß [Diet03] auch eine stark motivierende Wirkung. Der Einsatz von LEGO Mindstorms als Lernbehelf macht sich hier also voll bezahlt.

Es sollte jedoch klar sein, dass sich dieser Ansatz nicht für eine Einführung in die objektorientierten Konzepte eignet, da er bereits UML-Kenntnisse und damit verbunden entsprechendes Wissen hinsichtlich Objekten und Klassen voraussetzt. Sehr wohl kann diese Vorgehensweise aber zwecks Vertiefung im Schulunterricht eingesetzt werden.

8 Zusammenfassung und Bewertung

Es ist Zeit für eine Zusammenfassung sowie für ein Gesamtresümee in Bezug auf die Vermittlung der objektorientierten Konzepte mittels LEGO Mindstorms.

Kapitel 2 gab eine Einführung in die Didaktik der Informatik und verfolgte das Ziel, dem Leser die vielfältigen Möglichkeiten zur Unterrichtsgestaltung aufzuzeigen. Neben diversen Begriffserklärungen und der Betrachtung des Informatikunterrichts von geschichtlicher Seite her, wurde danach über die unterschiedlichen lerntheoretischen Konzepte, didaktischen Modelle und Prinzipien sowie Lehr- und Lernmethoden informiert. Hierbei zeigte sich, dass es heutzutage jede Menge interessante Alternativen zum Frontalunterricht gibt, die speziell in Zusammenhang mit dem Informatikunterricht interessant sind. Die zahlreichen Herausforderungen, denen sich der Lehrende im Informatikunterricht zu stellen hat, rundeten diesen Abschnitt ab.

Kapitel 3 schaffte die Grundlagen in Sachen Objektorientierung. Auch hier wurde wiederum ein Blick auf die geschichtliche Entwicklung geworfen, ehe die objektorientierten Konzepte erarbeitet wurden. Es konnte dabei festgestellt werden, dass jene nicht fix definiert sind – die wissenschaftliche Diskussion dazu ist immer noch im Gange – und dass sich diese von Werk zu Werk unterscheiden. Als Basiskonzepte wurden letztendlich Objekte, Klassen, der Polymorphismus und die Vererbung ausgewählt. Diese Selektion lässt sich im Nachhinein als sinnvoll bezeichnen, wenngleich die Beschäftigung mit den konkreten Ansätzen zeigte, dass das Prinzip des Polymorphismus aufgrund seiner Komplexität im objektorientierten Anfangsunterricht kaum eine Rolle spielt. Des Weiteren wurden in diesem Abschnitt der objektorientierte Entwicklungsprozess sowie die konkrete Umsetzung der Basiskonzepte in der Programmiersprache Java vorgestellt.

Kapitel 4 führte dann in LEGO Mindstorms ein. Die grundlegende Idee hinter diesen Roboterbaukästen, nämlich Kinder durch anfassbare Technik für selbige zu begeistern, wurde dem Leser dabei anhand der geschichtlichen Entwicklung nähergebracht. Im Anschluss daran wurde der Roboteraufbau für die NXT-Version erklärt. Für die weitere Arbeit war vor allem die Auseinandersetzung mit den unterschiedlichen Programmiermöglichkeiten eines solchen Roboters interessant. Im Konkreten wurde dabei auf die graphische Entwicklungsumgebung NXT-G und leJOS NXJ, das den Roboter mittels Java programmierbar macht, eingegangen. Die Beschäftigung mit ersterer Entwicklungsumgebung entpuppte sich im Nachhinein als überflüssig, denn dazu ließen sich im Gegensatz zu leJOS NXJ keine geeigneten Ansätze finden. Zum Abschluss des Abschnitts wurde noch über die vielfältigen Einsatzmöglichkeiten von LEGO Mindstorms im Ausbildungsbereich berichtet, wobei sich dabei zeigte, dass die Gruppenarbeit und das entdeckende Lernen gut mit den Roboterbaukästen harmonieren.

Kapitel 5 beschäftigte sich mit allgemeinen Themen, die in Verbindung mit der Vermittlung objektorientierter Konzepte stehen. Es ließ sich dabei feststellen, dass die Diskussion, ob im Programmierunterricht mit der Objektorientierung begonnen werden soll, nach wie vor andauert. Auch zeigte sich, dass das Vermitteln als auch das Erlernen des objektorientierten Paradigmas mit diversen Problemen verbunden ist. Zur Vermeidung dieser folgten Empfehlungen für die Vorgehensweise im Einsteigerunterricht und Tipps dazu. Ebenfalls wurden in diesem Abschnitt Kriterien zur

Bewertung von Anfängersystemen vorgestellt, die dann später zur Evaluierung von NXT-G und leJOS NXJ herangezogen wurden.

Kapitel 6 stellte im Anschluss daran verschiedene konkrete Ansätze zum Einführen von Schülern in die Objektorientierung und deren zugehörige Konzepte vor. Hierzu fanden sich in der Literatur jede Menge unterschiedliche Vorgehensweisen, die gemäß ihrem Schwerpunkt einer der folgenden Kategorien zugeteilt wurden: Objektspiele, Modellierung, Mikrowelten, Programmierumgebungen, Klassenbibliotheken und LEGO Mindstorms. In Verbindung mit den Roboterbaukästen wurden fünf mögliche Vorgehensweisen vorgestellt, die großteils auf einer der leJOS-Versionen und dadurch Java basierten und dann im folgenden Abschnitt hinsichtlich ihrer Tauglichkeit untersucht wurden.

Kapitel 7 untersuchte, inwieweit sich LEGO Mindstorms für die Vermittlung der Grundprinzipien der Objektorientierung eignet. Hier wurden generell die Vor- und Nachteile der Roboterbaukästen im Schulunterricht betrachtet, wobei gezeigt werden konnte, dass erstere überwiegen. Anschließend wurde für jedes der Basiskonzepte analysiert, ob LEGO Mindstorms für dessen Vermittlung einen positiven Beitrag leisten kann. Sowohl für das Objekt- und Klassenkonzept als auch für die Vererbung kann dies deutlich bejaht werden. Danach wurde die Eignung von NXT-G und leJOS NXJ für den objektorientierten Anfangsunterricht überprüft und dabei wurde festgestellt, dass sich die zweite Programmiermöglichkeit für eine Einführung sehr gut eignet und dass sich damit alle Basiskonzepte vermitteln lassen. Abschließend wurden noch die konkreten Ansätze des vorherigen Kapitels unter die Lupe genommen.

Resümierend und in Beantwortung der eingangs gestellten Fragen lässt sich festhalten, dass sich LEGO Mindstorms bestens zur Vermittlung der objektorientierten Konzepte im Schulunterricht eignet.

Als große Vorteile lassen sich im Allgemeinen die motivierende Wirkung der Roboterbaukästen auf die Lernenden und das direkte Feedback bei der Programmierung nennen. Weiters stellt LEGO Mindstorms eine skalierbare Unterrichtshilfe dar, die sich – ganz dem Alter der Schüler entsprechend – sehr flexibel einsetzen lässt.

In Zusammenhang mit der Objektorientierung helfen die realen, anfassbaren Roboterobjekte den Schülern deutlich dabei, ein Verständnis für den informatischen Objekt- und Klassenbegriff zu entwickeln. LEGO Mindstorms fungiert hier als Brückenbauer zwischen der realen und der virtuellen Welt – es lässt quasi die Wirklichkeit und das Modell miteinander verschmelzen. Da den meisten Schülern das Abstrahieren große Probleme bereitet, ist diese Eigenschaft von großer Bedeutung.

Nachteilig gegenüber anderen Ansätzen, wie z. B. Mikrowelten, sind jedoch die mit der Anschaffung verbunden Kosten und die Beschränktheit, insofern, dass die auf Basis der Roboterbaukästen durchgeführten Projekte räumlich an die Schule gebunden sind, wenn nicht jeder Schüler mit einem eigenen Roboterbaukasten ausgerüstet wurde. Aus didaktischer Sicht lassen sich allerdings unter Verwendung einer vernünftigen Programmiersprache und -umgebung keinerlei Nachteile feststellen.

Aus konzeptueller Sicht können mittels leJOS NXJ sämtliche objektorientierte Basiskonzepte – Objekte, Klassen, Polymorphimus und Vererbung – sinnvoll und verständlich vermittelt werden. Natürlich setzt dies aber die Wahl eines geeigneten Ansatzes voraus – hierzu wurden einige vorgestellt (Kapitel 6.6) und anschließend auch evaluiert (Kapitel 7.4). Man muss sich auch darüber im Klaren sein, dass man sich hierbei mit den Schülern in die Welt von Java begibt.

Zur Heranführung an dessen Syntax empfiehlt sich anfangs die Beistellung lauffähiger Programme, an denen die Lernenden erst einmal nur kleine Änderungen vornehmen müssen. Schrittweise können sie dann mit immer schwierigeren Aufgaben konfrontiert werden. Loslösen sollte man sich von dem Gedanken, dass die Schüler nach den wenigen Unterrichtseinheiten imstande sind, selbstständig Java-Programme schreiben zu können. Das sollte bei der Vermittlung der objektorientierten Konzepte auch nicht im Vordergrund stehen, vielmehr sollten die Unterrichteten die wichtigsten Prinzipien erfasst und verstanden sowie ein gutes Gefühl dafür entwickelt haben.

Die Beschäftigung mit der graphischen Programmieroberfläche NXT-G zeigte, dass sich diese nicht für eine Einführung in die grundlegenden Prinzipien der Objektorientierung eignet, da damit weder die Vererbung noch der Polymorphismus vermittelt werden kann.

Aufbauend auf dieser bzw. in Anlehnung an diese Arbeit lassen sich weitere Analysen durchführen. So könnte man andere, für die Programmierung von LEGO Mindstorms verfügbare Sprachen und Umgebungen hinsichtlich ihrer Einsatztauglichkeit im objektorientierten Anfangsunterricht einer kritischen Betrachtung unterziehen und jene dabei auch leJOS NXJ gegenüberstellen. Ebenso könnte man versuchen, auf Basis der vorgestellten konkreten Ansätze eine neue Vorgehensweise zu entwickeln, welche die festgestellten Schwächen ausmerzt. Natürlich sollten diese dann auch konkret im Schulunterricht getestet und evaluiert werden.

Abschließend sei angemerkt, dass – wie sooft im Leben – auch bei der Vermittlung der objektorientierten Konzepte viele Wege ans Ziel führen. Gleichgültig, für welchen Ansatz sich der Lehrer letztendlich entscheidet, unter Berücksichtigung des eingangs erwähnten Mottos "Spiel, Spaß und was zum Lernen" stehen die Chancen um ein Vielfaches besser, dass sich das Erlernte auch tatsächlich in der Schülerköpfen verankert.

9 Literaturverzeichnis

9.1 Publikationen

[Altm05] Aender Altman: *Didaktische Konzepte von Lernsoftware – Konstruktionismus und LEGO MINDSTORMS*. Diplomarbeit, Technische Universität Wien, 2005.

[Asto07] Dave Astolfo, Mario Ferrari, Giulio Ferrari: *Building Robots with LEGO Mindstorms NXT*. Syngress Publishing, 2007.

[Bagn07] Brian Bagnell: *Maximum Lego NXT – Building Robots with Java Brains*. Variant Press, 2007.

[Barn02] David J. Barnes: *Teaching Introductory Java through LEGO MINDSTORMS Models*. In: ACM SIGCSE Bulletin, 34 (1), S. 147-151, 2002.

[Barn06] David J. Barnes, Michael Kölling: *Java lernen mit BlueJ – Eine Einführung in die objektorientierte Programmierung* (3. Auflage). Pearson Studium Verlag, 2006.

[Baum94] Peter Baumgartner, Sabine Payr: *Lernen mit Software*. Studien Verlag, 1994.

[Baum96] Rüdeger Baumann: *Didaktik der Informatik* (2. Auflage). Ernst Klett Verlag, 1996.

[Beck01] Byron W. Becker: *Teaching CS1 with Karel the Robot in Java*. In: ACM SIGCSE Bulletin, 33 (1), S. 50-54, 2001.

[Bena08] Tamar Benaya, Ela Zur: *Understanding Object Oriented Programming Concepts in an Advanced Programming Course*. In: *ISSEP '08: Proceedings of the 3rd international conference on Informatics in Secondary Schools – Evolution and Perspectives*, S. 161-170, 2008.

[Bole04] Dietrich Boles, Cornelia Boles: *Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell*. B. G. Teubner Verlag, 2004.

[Bole05] Dietrich Boles: *Spielerisches Erlernen der Programmierung mit dem Java-Hamster-Modell*. In: Steffen Friedrich (Hrsg.): *Unterrichtskonzepte für informatische Bildung, INFOS 2005, 11. GI-Fachtagung Informatik und Schule, 28.-30. September 2005 an der TU Dresden*, S. 243-252, 2005.

[Börs07] Jürgen Börstler: *Objektorientiertes Programmieren – Machen wir irgendwas falsch?*. In: Sigrid E. Schubert (Hrsg.): *Didaktik der Informatik in Theorie und Praxis. INFOS 2007: 12. GI-Fachtagung Informatik und Schule, 19.-21. September 2007 in Siegen*, S. 9-20, 2007.

[Brin04] Torsten Brinda: *Didaktisches System für objektorientiertes Modellieren im Informatikunterricht der Sekundarstufe II*. Dissertation, Universität Siegen, 2004.

[Broo95] Frederick P. Brooks: *The Mythical Man-Month – Essays on Software Engineering* (20. Auflage). Addison-Wesley Longman Publishing, 1995.

[Budd91] Timothy Budd: *An Introduction to Object-Oriented Programming*. Addison-Wesley Longman Publishing, 1991.

[Crut95] Cecile K. M. Crutzen, Hans-Werner Hein: *Objektorientiertes Denken als didaktische Basis der Informatik*. In: Sigrid E. Schubert (Hrsg.): *Innovative Konzepte für die Ausbildung, 7. GI-Fachtagung Informatik und Schule, INFOS'95, Chemnitz, 25.-28. September 1995*, S. 49-158, 1995.

[Dagd05] Vassilios Dagdilelis, Maya Sartatzemi, Katerina Kagani: *Teaching (with) Robots in Secondary Schools: some new and not-so-new Pedagogical problems*. In: *ICALT '05: Proceedings of the Fifth IEEE International Conference on Advanced Learning Technologies*, S. 757-761, 2005.

[Deck94] Rick Decker, Stuart Hirshfield: *The Top 10 Reasons Why Object-Oriented Programming Can't Be Taught in CS1*. In: ACM SIGCSE Bulletin, 26 (1), S. 51-55, 1994.

[Diet01] Ruth Dietzel, Tim Rinkens: *Eine Einführung in die Objektorientierung mit Lego Mindstorms Robotern – Erfahrungsbericht aus dem Unterricht*. In: Reinhard Keil-Slawik, Johannes Magenheim (Hrsgg.): *Informatikunterricht und Medienbildung, INFOS 2001, 9. GI-Fachtagung Informatik und Schule, 17.-20. September 2001 in Paderborn*, S. 193-199, 2001.

[Diet03] Ira Diethelm, Leif Geiger, Albert Zündorf: *Fujaba goes Mindstorms: Objektorientierte Modellierung zum Anfassen*. In: Peter Hubwieser (Hrsg.): *Informatische Fachkonzepte im Unterricht, INFOS 2003, 10. GI-Fachtagung Informatik und Schule, 17.-19. September 2003 in Garching bei München*, S. 225-235, 2003.

[Diet05a] Ira Diethelm, Leif Geiger, Albert Zündorf: *Mit Klebezettel und Augenbinde durch die Objektwelt*. In: Steffen Friedrich (Hrsg.): *Unterrichtskonzepte für informatische Bildung, INFOS 2005, 11. GI-Fachtagung Informatik und Schule, 28.-30. September 2005 an der TU Dresden*, S. 149-160, 2005.

[Diet05b] Ira Diethelm, Leif Geiger, Albert Zündorf: *Rettet Prinzessin Ada: Am leichtesten objektorientiert*. In: Steffen Friedrich (Hrsg.): *Unterrichtskonzepte für informatische Bildung, INFOS 2005, 11. GI-Fachtagung Informatik und Schule, 28.-30. September 2005 an der TU Dresden*, S. 161-172, 2005.

[Diet07] Ira Diethelm: *Strictly models and objects first – Unterrichtskonzept für objektorientierte Modellierung*. In: Sigrid E. Schubert (Hrsg.): *Didaktik*

der Informatik in Theorie und Praxis. INFOS 2007: 12. GI-Fachtagung Informatik und Schule, 19.-21. September 2007 in Siegen, S. 45-56, 2007.

[Dißm03] Stefan Dißmann: *Die Einführung objektorientierter Gestaltungsprinzipien anhand von Rollenspielen*. In: Johannes Siedersleben, Debora Weber-Wulff (Hrsgg.): *Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin 2003*, S. 30-40, 2003.

[Feld88] Richard M. Felder, Linda K. Silverman: *Learning and Teaching Styles in Engineering Education*. In: Engineering Education, 78 (7), S. 674-681, 1988.

[Flow02] Thomas R. Flowers, Karl A. Gossett: *Teaching Problem Solving, Computing, and Information Technology with Robots*. In: Journal of Computing Sciences in Colleges, 17 (6), S. 45-55, 2002.

[Frei02] Ulli Freiberger: *Karel – Eine Übersicht über verschiedene Entwicklungen, die auf der Idee von "Karel, the Robot" basieren*. Arbeitspapier, Luitpold-Gymnasium München, 2002.

[Gall95] Harald Gall, Manfred Hauswirth, René Klösch: *Objektorientierte Konzepte in Smalltalk, C++, Objective-C, Eiffel und Modula-3*. In: Informatik Spektrum, 18 (4), S. 195-202, 1995.

[Gran05] Roar Granerud: *Teaching object oriented programming to youths using the control technology Lego Mindstorms*. Diplomarbeit, Universität Oslo, 2005.

[Gran06] Roar Granerud, Jens Kaasbøll, Richard Borge, Christian Holmboe, Ole Smørdal: *Children's understanding of object-orientation*. In: Annita Fjuk, Amela Karahasanović, Jens Kaasbøll (Hrsgg.): *Comprehensive Object-Oriented Learning: The Learner's Perspective*, S. 27-47, 2006.

[Hadj98] Said Hadjerrouit: *Java as First Programming Language: A Critical Evaluation*. In: ACM SIGCSE Bulletin, 30 (2), S. 43-47, 1998.

[Hart07] Werner Hartmann, Michael Näf, Raimond Reichert: *Informatikunterricht planen und durchführen* (1. Nachdruck). Springer Verlag, 2007.

[Heub04] Alfred Heubaum: *Möglichkeiten und Grenzen maschineller Intelligenz*. In: LOG IN, 128/129, S. 62-79, 2004.

[Hirs03] Anthony Hirst, Jeffrey Johnson, Marian Petre, Blaine A. Price, Mike Richards: *What is the best programming environment/language for teaching robotics using Lego Mindstorms?*. In: Artificial Life and Robotics, 7 (3), S. 124-131, 2003.

[Holl97] Simon Holland, Robert Griffiths, Mark Woodman: *Avoiding Object Misconceptions*. In: ACM SIGCSE Bulletin, 29 (1), S. 131-134, 1997.

[Hubw05] Peter Hubwieser: *Von der Funktion zum Objekt – Informatik für die Sekundarstufe I*. In: Steffen Friedrich (Hrsg.): *Unterrichtskonzepte für informatische Bildung, INFOS 2005, 11. GI-Fachtagung Informatik und Schule, 28.-30. September 2005 an der TU Dresden*, S. 27-41, 2005.

[Hubw07] Peter Hubwieser: *Didaktik der Informatik – Grundlagen, Konzepte, Beispiele* (3. Auflage). Springer Verlag, 2007.

[Humb06] Ludger Humbert: *Didaktik der Informatik mit praxiserprobtem Unterrichtsmaterial* (2. Auflage). B. G. Teubner Verlag, 2006.

[Jank02] Werner Jank, Hilbert Meyer: *Didaktische Modelle* (5. Auflage). Cornelsen Scriptor Verlag, 2002.

[Kohl06] Lutz Kohl, Ralf Romeike: *Aktueller Stand der Objektorientierung bei Informatiklehrerinnen und -lehrern*. In: Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): *HDI 2006: Hochschuldidaktik der Informatik – Organisation, Curricula, Erfahrungen. 2. GI-Fachtagung 7.-8. Dezember 2006 in München, Germany*, S. 63-75, 2006.

[Köll99a] Michael Kölling: *The problem of teaching object-oriented programming – Part I: Languages*. In: Journal of Object-Oriented Programming, 11 (8), S. 8-15, 1999.

[Köll99b] Michael Kölling: *The problem of teaching object-oriented programming – Part II: Environments*. In: Journal of Object-Oriented Programming, 11 (9), S. 6-12, 1999.

[Köll01] Michael Kölling, John Rosenberg: *Guidelines for Teaching Object Orientation with Java*. In: ACM SIGCSE Bulletin, 33 (3), S. 33-36, 2001.

[Korn04] Friedrich W. Korn: *Grundwissen Didaktik* (4. Auflage). Ernst Reinhardt Verlag, 2004.

[Kure06] Shuji Kurebayashi, Toshiyuki Kamada, Susumu Kanemune: *Learning Computer Programming with Autonomous Robots*. In: Roland Mittermeir (Hrsg.): *Informatics Education – The Bridge between Using and Understanding Computers, International Conference in Informatics in Secondary Schools – Evolution and Perspectives, ISSEP 2006, Vilnius, Lithuania, November 7-11, 2006, Proceedings*, S. 138-149, 2006.

[Lawh02] Pamela B. Lawhead, Constance G. Bland, David J. Barnes, Michaele E. Duncan, Michael Goldweber, Ralph G. Hollingsworth, Madeleine Schep: *A Road Map for Teaching Introductory Programming Using LEGO Mindstorms Robots*. In: ACM SIGCSE Bulletin, 35 (2), S. 191-201, 2002.

[Lewi00] John Lewis: *Myths about Object-Orientation and Its Pedagogy*. In: ACM SIGCSE Bulletin, 32 (1), S. 245-249, 2000.

[Mcna06] Myles McNally, Michael Goldweber, Barry Fagin, Frank Klassner: *Do Lego MindStorms Robots have a Future in CS Education?*. In: ACM SIGCSE Bulletin, 38 (1), S. 61-62, 2006.

[Meye04] Matthias Meyer, Lothar Wendehals: *Teaching Object-Oriented Concepts with Eclipse*. In: *eclipse '04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, S. 1-5, 2004.

[Noll07] Werner Noll: *Einführung in die OOP mit BlueJ*. Examensarbeit, Goethe-Universität Frankfurt/Main, 2007.

[Perr98] Corrina Perrone, Alexander Repenning: *Graphical Rewrite Rule Analogies: Avoiding the Inherit or Copy & Paste Reuse Dilemma*. In: *Proceedings of the IEEE Symposium on Visual Languages*, S. 40-46, 1998.

[Poet09] Arnd Poetzsch-Heffter: *Konzepte objektorientierter Programmierung – Mit einer Einführung in Java* (2. Auflage). Springer Verlag, 2009.

[Pill09] Nelishia Pillay: *A Study of Object-Oriented Design Errors Made by Novice Programmers*. In: *SACLA '09: Proceedings of the 2009 Annual Conference of the Southern African Computer Lecturers' Association*, S. 101-104, 2009.

[Prei08] Andreas Prein: *Didaktische Entwürfe zum Pflichtgegenstand "Angewandte Programmierung" des ersten Jahrganges einer HTL für Informationstechnologie*. Diplomarbeit, Technische Universität Wien, 2008.

[Punt08] Franz Puntigam: *Objektorientierte Programmierung*. Skriptum zur Lehrveranstaltung, Technische Universität Wien, 2008.

[Reic05] Raimond Reichert, Jürg Nievergelt, Werner Hartmann: *Programmieren mit Kara – Ein spielerischer Zugang zur Informatik* (2. Auflage). Springer Verlag, 2007.

[Rink01] Tim Rinkens: *Einführung in die objektorientierte Programmierung in der gymnasialen Oberstufe mit Hilfe einer Java VM im Lego Mindstorms RCX*. Examensarbeit, Universität Paderborn, 2001.

[Rome07] Ralf Romeike: *Kriterien kreativen Informatikunterrichts*. In: Sigrid E. Schubert (Hrsg.): *Didaktik der Informatik in Theorie und Praxis. INFOS 2007: 12. GI-Fachtagung Informatik und Schule, 19.-21. September 2007 in Siegen*, S. 57-68, 2007.

[Sart05] Maya Sartatzemi, Vassilios Dagdilelis, Katerina Kagani: *Teaching Programming with Robots: A Case Study on Greek Secondary Education*. In: Panayiotis Bozanis, Elias N. Houstis (Hrsgg.): *Advances in Informatics, 10th Panhellenic Conference on Informatics, PCI 2005, Volos, Greece, November 11-13, 2005, Proceedings*, S. 502-512, 2005.

[Schm09] Peter Schmidt: *Einsatz von Lego Mindstorms im Unterricht der HTL für Informationstechnologie*. Diplomarbeit, Technische Universität Wien, 2009.

[Schu03] Carsten Schulte: *Lehr-Lernprozesse im Informatik-Anfangsunterricht – Theoriegeleitete Entwicklung eines Unterrichtskonzepts zur Objektorientierung in der Sekundarstufe II*. Dissertation, Universität Paderborn, 2003.

[Schu04] Sigrid E. Schubert, Andreas Schwill: *Didaktik der Informatik*. Spektrum Akademischer Verlag, 2004

[Schu08] Torsten Schultz: *Entwicklung einer handlungsorientierten Einführung in die objektorientierte Analyse und Modellierung im Anfangsunterricht der Jahrgangsstufe 11 durch Objekt-Rollenspiele*. Examensarbeit, Seminar Gymnasium/Gesamtschulen Hamm, 2008.

[Schw95] Andreas Schwill: *Programmierstile im Anfangsunterricht*. In: Sigrid E. Schubert (Hrsg.): *Innovative Konzepte für die Ausbildung, 7. GI-Fachtagung Informatik und Schule, INFOS'95, Chemnitz, 25.-28. September 1995*, S. 178-187, 1995.

[Smit93] David N. Smith: *Konzepte der objektorientierten Programmierung*. McGraw-Hill Verlag, 1993.

[Spol95] Siegfried Spolwig: *Objektorientierte Programmierung im Anfangsunterricht*. In: LOG IN, 15 (3), S. 43-49, 1995.

[Spol98] Siegfried Spolwig: *"Hello World" in OOP*. In: LOG IN, 19 (5), S. 38-42, 1998.

[Stol07] Hans Stolzlechner: *Forschendes Lernen – Ein alternativer Weg zum Erlernen einer Programmiersprache*. Arbeitspapier, Polytechnische Schule Tamsweg, 2007.

[Weic05] Nicole Weicker, Karsten Weicker: *Didaktische Anmerkungen zur Unterstützung der Programmierlehre durch E-Learning*. In: Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsgg.): *DeLFI 2005: 3. Deutsche e-Learning Fachtagung Informatik, der Gesellschaft für Informatik e.V. (GI) 13.-16. September 2005 in Rostock*, S. 435-446, 2005.

[Whit07] Alma K. Whitfield, Stewart Blakeway, George E. Herterich, Chris Beaumont: *Programming, disciplines and methods adopted at Liverpool Hope University*. In: ITALICS, 6 (4), S. 145-168, 2007.

[Witt92] Kurt-Ulrich Witt: *Einführung in die objektorientierte Programmierung*. R. Oldenburg Verlag, 1992.

[Wood97] Mark Woodman, Andrew Law, Simon Holland, Robert Griffiths: *The Object Shop – Using CD-ROM Multimedia To Introduce Object Concepts*. In: ACM SIGCSE Bulletin, 29 (1), S. 345-349, 1997.

9.2 Webseiten

[@csis] http://www.csis.pace.edu/~bergin/papers/Whynotproceduralfirst.html
(zuletzt am 15.10.2009 besucht)

[@deba] http://www.debacher.de/wiki/NXT-G
(zuletzt am 15.10.2009 besucht)

[@hand] http://www.hands-on-technology.de
(zuletzt am 15.10.2009 besucht)

[@leara] http://www.learn-line.nrw.de/angebote/oop/medio/didaktik/grundlagen/warum.html
(zuletzt am 15.10.2009 besucht)

[@learb] http://www.learn-line.nrw.de/angebote/oop/medio/theorie/programmierung
(zuletzt am 15.10.2009 besucht)

[@learc] http://www.learn-line.nrw.de/angebote/oop/medio/sum
(zuletzt am 15.10.2009 besucht)

[@lego] http://www.legorobotik.ch
(zuletzt am 15.10.2009 besucht)

[@lejoa] http://lejos.sourceforge.net/nxt/nxj/tutorial/Preliminaries/Intro.htm
(zuletzt am 15.10.2009 besucht)

[@lejob] http://lejos.sourceforge.net/nxt/nxj/tutorial/Preliminaries/GettingStarted.htm
(zuletzt am 15.10.2009 besucht)

[@meth] http://methodenpool.uni-koeln.de/frameset_uebersicht.htm
(zuletzt am 15.10.2009 besucht)

[@oszha] http://oszhdl.be.schule.de/gymnasium/faecher/informatik/softwareprojekte/warum_oop.htm
(zuletzt am 15.10.2009 besucht)

[@oszhb] http://oszhdl.be.schule.de/gymnasium/faecher/informatik/ooa-ood/ooa_ziele.htm
(zuletzt am 15.10.2009 besucht)

[@wikia] http://de.wikipedia.org/wiki/Johann_Amos_Comenius
(zuletzt am 15.10.2009 besucht)

[@wikib] http://de.wikipedia.org/wiki/Wolfgang_Klafki
(zuletzt am 15.10.2009 besucht)

[@wikic] http://de.wikipedia.org/wiki/Karl_Steinbuch
(zuletzt am 15.10.2009 besucht)

[@wikid] http://de.wikipedia.org/wiki/Ruth_Cohn
(zuletzt am 15.10.2009 besucht)

[@wikie] http://de.wikipedia.org/wiki/Iwan_Petrowitsch_Pawlow
(zuletzt am 15.10.2009 besucht)

[@wikif] http://de.wikipedia.org/wiki/John_B._Watson
(zuletzt am 15.10.2009 besucht)

[@wikig] http://de.wikipedia.org/wiki/Burrhus_Frederic_Skinner
(zuletzt am 15.10.2009 besucht)

[@wikih] http://de.wikipedia.org/wiki/Behaviorismus
(zuletzt am 15.10.2009 besucht)

[@wikii] http://de.wikipedia.org/wiki/Kognitivismus
(zuletzt am 15.10.2009 besucht)

[@wikij] http://de.wikipedia.org/wiki/Donald_O._Hebb
(zuletzt am 15.10.2009 besucht)

[@wikik] http://de.wikipedia.org/wiki/Konstruktivismus_(Lernpsychologie)
(zuletzt am 15.10.2009 besucht)

[@wikil] http://de.wikipedia.org/wiki/Curriculare_Didaktik
(zuletzt am 15.10.2009 besucht)

[@wikim] http://de.wikipedia.org/wiki/Konstruktivistische_Didaktik
(zuletzt am 15.10.2009 besucht)

[@wikin] http://de.wikipedia.org/wiki/Didaktik
(zuletzt am 15.10.2009 besucht)

[@wikio] http://de.wikipedia.org/wiki/Frederick_P._Brooks
(zuletzt am 15.10.2009 besucht)

[@wikip] http://de.wikipedia.org/wiki/Objektorientierte_Programmiersprache
(zuletzt am 15.10.2009 besucht)

[@wikiq] http://de.wikipedia.org/wiki/Liste_objektorientierter_Programmiersprachen
(zuletzt am 15.10.2009 besucht)

[@wikir] http://de.wikipedia.org/wiki/Lego_Mindstorms
(zuletzt am 15.10.2009 besucht)

[@wikis] http://de.wikipedia.org/wiki/Seymour_Papert
(zuletzt am 15.10.2009 besucht)

[@wikt] http://de.wiktionary.org/wiki/Roboter
(zuletzt am 15.10.2009 besucht)