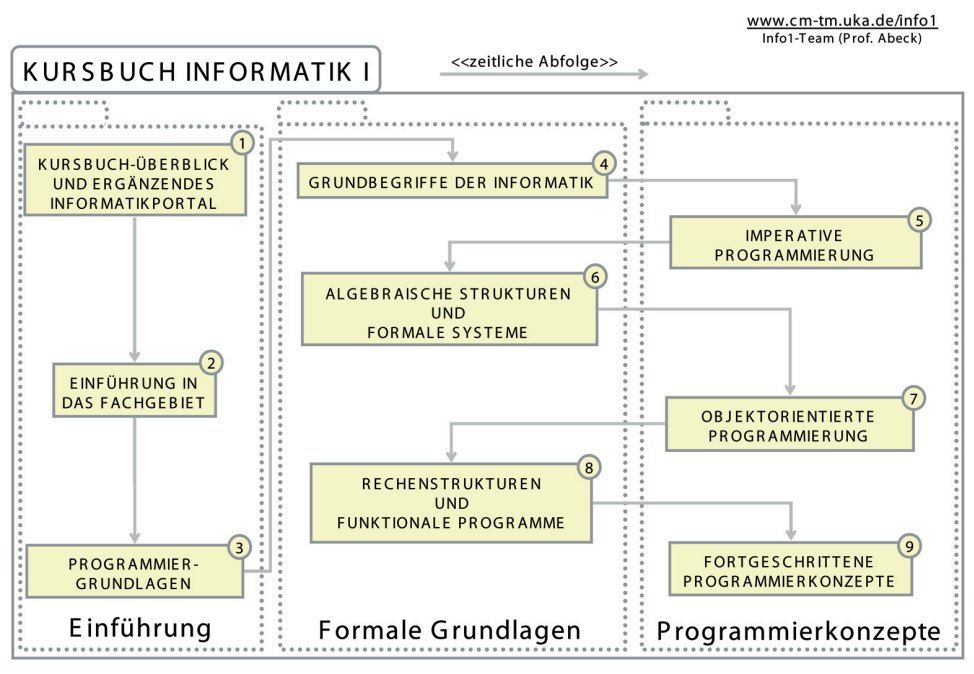
Sebastian Abeck

KURSBUCH INFORMATIK I

Formale Grundlagen der Informatik und Programmierkonzepte am Beispiel von Java



universitätsverlag karlsruhe

Sebastian Abeck

KURSBUCH INFORMATIK I

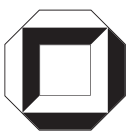
Universität Karlsruhe (TH) – Fakultät für Informatik –
Institut für Telematik – Cooperation & Management (Prof. Abeck) –
Zirkel 2 – 76128 Karlsruhe
http://www.cm-tm.uka.de/info1
abeck@cm-tm.uka.de

KURSBUCH INFORMATIK I

Formale Grundlagen der Informatik und
Programmierkonzepte am Beispiel von Java

Eine Einführung in die Informatik auf der Grundlage
etablierter Lehrbücher

von
Sebastian Abeck



universitätsverlag karlsruhe

Impressum

Universitätsverlag Karlsruhe
c/o Universitätsbibliothek
Straße am Forum 2
D-76131 Karlsruhe
www.uvka.de

Universitätsverlag Karlsruhe 2005
Print on Demand

ISBN 3-937300-68-6

VORWORT

Informatik ist ein Kunstwort, gebildet aus Information und in Analogie zu Mathematik[1]. **Information** ist der zentrale Untersuchungsgegenstand dieses noch relativ jungen Fachgebiets; die **Mathematik** liefert die zwingend erforderlichen formalen Methoden, um Fragestellungen der **Verarbeitung, Speicherung und Übertragung von Information** zu durchdringen. Da der Information in der heutigen Informationsgesellschaft und morgigen Wissensgesellschaft eine immer größere Bedeutung zukommt, ist die **Informatik eine Schlüsselwissenschaft des 21. Jahrhunderts**.

In insgesamt **9 Kurseinheiten** wird durch das vorliegende Kursbuch in das spannende Gebiet der Informatik eingeführt. Die inhaltlichen Schwerpunkte liegen im **algorithmischen Denken** und im **Programmieren**. Es werden zum einen die wichtigsten **theoretischen Grundlagen** der Informatik behandelt, zum anderen werden **praktische Fähigkeiten** vermittelt, die zur selbstständigen Erstellung von gut geschriebenen Programmen auf der Basis der höheren Programmiersprache **Java** und der Modellierungssprache ***Unified Modeling Language*** (UML) erforderlich sind.
Das Kursbuch wendet sich an Studierende, die am Anfang ihres Informatikstudiums stehen. Die Kurseinheiten bilden die Grundlage einer Lehrveranstaltung, die ca. **60 Vorlesungsstunden** (à 45 Minuten) umfasst. Aufgrund des **modularisierten Aufbaus** lassen sich aus dem Kursbuch die folgenden drei **Teilkurse** zusammenstellen: (1) **Einführung in die Informatik** (2) **Formale Grundlagen** und (3) **Programmierkonzepte am Beispiel von Java**. Jede Kurseinheit ist durch ein **ausführlich ausgearbeitetes Kursdokument** beschrieben. In die Dokumente sind die vom Dozenten in der Vorlesung verwendeten Folien eingebunden. Dadurch erhält der Studierende[2] ein optimal auf die Vorlesung zugeschnittenes Skript.

Dem Kursbuch liegen drei etablierte deutschsprachige Informatik-Lehrbücher zugrunde. Die Inhalte zur Einführung und zu den formalen Grundlagen orientieren sich an den Lehrbüchern von Gerhard Goos (Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer-Verlag, 1997) und Manfred Broy (Informatik – Eine grundlegende Einführung, Band 1: Programmierung und Rechenstrukturen, Springer Verlag, 1998). Die Programmierkonzepte werden in Anlehnung an das Lehrbuch von Hanspeter Mössenböck (Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag, 2003) vermittelt. Die Lehrbücher sollten zur Ergänzung und zur Vertiefung der in den entsprechenden Kurseinheiten präsentierten Inhalte genutzt werden. Das Kursbuch ist **kein Ersatz für die genannten Lehrbücher**. Insbesondere sei betont, dass **im Kursbuch gewisse Inhalte bewusst ausgeblendet wurden**, da Ergänzungen in Form von Tafelbildern durch den Dozenten während der Lehrveranstaltung erfolgen. Daher ist das Kursbuch immer in Verbindung mit dem nachfolgend näher beschriebenen Informatikportal zu sehen, über das ergänzende Informationen (z.B. Aufzeichnungen von Lehrveranstaltungen) über das Internet zur Verfügung gestellt werden. In dieser Kombination stellt das Kursbuch eine neue und moderne Form eines multimedialen Lernangebots bereit.

In den Kursdokumenten sind **Interaktionsfolien** enthalten, durch die der Lernende zum **aktiven Mitdenken** eingeladen wird. In einer Interaktionsfolie werden Aufgaben gestellt, deren Lösung vom Dozenten gemeinsam mit den Teilnehmern der Vorlesungsveranstaltung anhand eines **Tafelbildes** erarbeitet wird. In den im Wintersemester 04/05 und 05/06 an der Universität Karlsruhe (TH) durchgeführten Informatik-Einführungsvorlesungen verwendet der Dozent eine (mittels eines *Tablet-PC* realisierte) **elektronische Tafel**. Hierdurch ist es möglich, das

[1] Peter Rechenberg: Was ist Informatik? – Carl Hanser Verlag, 1994.
[2] Die im Kursbuch verwendeten Bezeichnungen sind geschlechtsneutral zu verstehen.

Tafelbild und die Folienannotationen gemeinsam mit dem gesprochenen Wort des Dozenten in Form eines *Screen-Capture*-Videos aufzuzeichnen. Um eine pädagogisch sinnvolle Nutzung des **Vorlesungsvideos** zu gewährleisten, ist das Video in das zugehörige Kursdokument so integriert, dass zu jeder in der Vorlesung präsentierten Folie der zugehörige Videoausschnitt zugegriffen werden kann. Alle notwendigen Informationen zur Verwendung der in der Forschungsgruppe von Prof. Abeck entwickelten so genannten **Living Documents** (LDocs) finden sich im ersten Kursdokument KURSBUCH-ÜBERBLICK UND ERGÄNZENDES INFORMATIKPORTAL. Die LDocs bieten für Interessierte innerhalb und außerhalb der Universität eine attraktive Möglichkeit, die **Vorlesungsveranstaltungen virtuell zu besuchen**. Das Portal ist über die Web-Adresse **http://www.cm-tm.uka.de/info1** zugänglich.

Die INFORMATIK-I-Veranstaltung ist das Ergebnis einer **Teamarbeit** zahlreicher, engagierter Angehörige der Fakultät für Informatik der Universität Karlsruhe (TH). Neben den Mitarbeitern der Forschungsgruppe Cooperation & Management (C&M) – hier ist insbesondere der Übungsleiter Dipl.-Inform. Karsten Krutz hervorzuheben – übernehmen mehrere Mitarbeiter der zentralen Einrichtungen der Fakultät wichtige Aufgaben bei der Durchführung der Veranstaltung. Insbesondere gehören zum **Info1-Team** auch zahlreiche Studierende, die als Tutoren tätig sind oder in Form von Diplomarbeiten, Studienarbeiten oder Praktika maßgeblich an der Bereitstellung des oben erwähnten Informatikportals und der LDocs beitragen.

Ich möchte mich bei allen Mitgliedern des Info1-Teams herzlich für ihre wertvollen Beiträge bedanken und wünsche allen Lesern dieses Kursbuchs viel Freude bei der Bearbeitung der Inhalte.

Karlsruhe, im August 2005 Sebastian Abeck

INHALTSVERZEICHNIS

Formale Grundlagen 99

Programmierkonzepte 247

Kursblock

EINFÜHRUNG

Im ersten Kursblock wird ein Überblick über den Aufbau und den Inhalt des Kursbuchs gegeben und das begleitend zum Kursbuch zur Verfügung stehende Informatikportal sowie die darüber abrufbaren multimedial aufbereiteten Lehrmaterialien werden vorgestellt. In den zwei folgenden Kurseinheiten wird ein erstes Grundverständnis des Fachgebiets der Informatik vermittelt und die Grundlagen der Programmierung am Beispiel der höheren Programmiersprache Java werden eingeführt.

Der Kursblock besteht aus den folgenden Kurseinheiten:

KURSBUCH-ÜBERBLICK UND ERGÄNZENDES INFORMATIKPORTAL

Kurzbeschreibung

Das Kursdokument beschreibt den Aufbau und den Inhalt des Kursbuches. Außerdem werden das ergänzend zum Kursbuch angebotene INFORMATIK-I-Portal (IPO) sowie die über das IPO bereitgestellten Living Documents (LDocs) vorgestellt.

Schlüsselwörter

(Web-) Portal, INFORMATIK-I-Portal, IPO, IPO-Dienstangebot, Aufbau, Oberfläche, Bedienung, Registrierung, Diskussionsforum, Living Document, LDoc, C&M-konformer Kurs, Windows, Linux, *Media Player*, *Codec*

Lernziele

1. Der inhaltliche Aufbau des Kursbuchs kann nachvollzogen werden.
2. Auf das Informatikportal kann zugegriffen werden und der Vorgang der Registrierung beim Portal wird verstanden.
3. Das Vorgehen zur Nutzung der Living Documents ist bekannt und die erforderliche Software (Windows oder Linux) kann auf dem eigenen Rechner installiert werden.

Hauptquellen

- Sebastian Abeck, Markus Gebhard, Karsten Krutz, Klaus Scheibenberger, Niko Schmid: Ein Portal zur Lehrunterstützung, Arbeitskonferenz "Elektronische Geschäftsprozesse", Klagenfurt, 2004.
- Sebastian Abeck, Pascal Bihler, Karsten Krutz, Christian Mayerl, Mads Stavang, Marco Willsch, C&M-konformer Kurs und Living Document, GI-Jahrestagung, Ulm, 2003.

Inhaltsverzeichnis

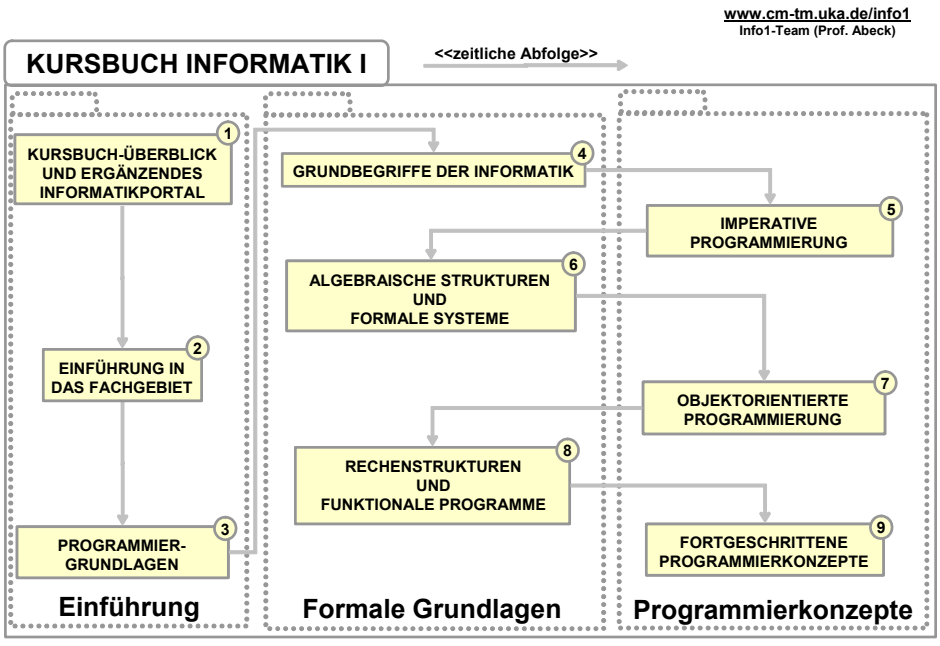
www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- AUFBAU DES KURSBUCHS UND DER KURSEINHEITEN
  - Kursblöcke, zeitlicher Ablauf, einheitliche Strukturierung jeder Kurseinheit
- INFORMATIK-I-PORTAL (IPO)
  - Einführung, Zugang und Bedienung, Registrierung und Teilnahme am Diskussionsforum
- LIVING DOCUMENTS (LDOCS)
  - Motivation, C&M-konformer Kurs, LDoc-Tools, Installation des Codec, Herunterladen zur Offline-Nutzung, Empfehlungen zur Nutzung

Information 1: KURSBUCH-ÜBERBLICK UND ERGÄNZENDES INFORMATIKPORTAL

1 AUFBAU DES KURSBUCHS UND DER KURS-DOKUMENTE

Das Kursbuch ist in neun Kurseinheiten aufgeteilt, zu denen jeweils ein einheitlich aufgebautes Kursdokument besteht. Das vorliegende Dokument ist das erste Kursdokument des Kursbuchs.



Information 2: AUFBAU DES KURSBUCHS - Überblick über die Kurseinheiten

Die Inhalte zerfallen in insgesamt 3 Kursblöcke (Einführung, Formale Grundlagen, Programmierkonzepte) mit jeweils 3 Kurseinheiten, wie das Diagramm in Information 2 zeigt. Durch die gerichteten Pfeile und die Nummerierung der Kurseinheiten wird aufgezeigt, in welcher Reihenfolge die Kurseinheiten in der Vorlesung behandelt werden. Der Inhalt einer

Kurseinheit wird üblicherweise innerhalb von 2 bis 4 Vorlesungsterminen (1,5 bis 3 Zeitstunden) behandelt.

Ein Schwerpunkt besteht darin, dem Studierenden das theoretische Rüstzeug der Informatik zu vermitteln (Kursblock Formale Grundlagen), durch die er einen fundierten Zugang zur Programmierung und zur Entwicklung von Informatiksystemen erhält (Kursblock Programmierkonzepte).

1.1 Strukturierung des Kursbuchs in Kursblöcke

Nachfolgend werden die Kursblöcke und die darin auftretenden Kurseinheiten kurz beschrieben.

1.1.1 Einführung

Es wird ein Überblick über den Aufbau und den Inhalt des KURSBUCHS INFORMATIK I gegeben. Die neben dem Kursbuch zur Verfügung gestellten Unterstützungssysteme in Form eines Informatikportals sowie multimedial aufbereiteter Lehrmaterialien werden eingeführt. Außerdem wird ein erstes Grundverständnis des Fachgebiets vermittelt.

- KURSBUCH-ÜBERBLICK UND ERGÄNZENDES INFORMATIKPORTAL ist das vorliegende Dokument.
- EINFÜHRUNG IN DAS FACHGEBIET zeigt die Teilgebiete der Informatik auf, beschreibt das Berufsbild eines Informatikers und stellt exemplarisch ein Informatiksystem als den zentralen Untersuchungsgegenstand des Fachgebiets vor.
- PROGRAMMIERGRUNDLAGEN umfassen die elementaren Fähigkeiten, um erste vollständige und lauffähige Programme zu erstellen.

1.1.2 Formale Grundlagen

Es wird die Theorie der Informatik soweit vermittelt, dass das mathematische Rüstzeug, das zur Programmierung im Kleinen erforderlich ist, vorhanden ist.

- GRUNDBEGRIFFE DER INFORMATIK wie Information, Modell, Algorithmus und Architektur sowie die damit verknüpften Konzepte und Theorien werden eingeführt und anhand von Beispielen präzisiert.
- ALGEBRAISCHE STRUKTUREN UND FORMALE SYSTEME liefern die mathematische Basis, auf der der Kern der theoretischen Informatik in Form der formalen Systeme aufbaut. Als algebraische Strukturen werden Halbgruppen, Relationen, Graphen und die Boolesche Algebra eingeführt. Die behandelten formalen Systeme sind Semi-Thue-Systeme, Markov-Algorithmen, Chomsky-Grammatiken und endliche Automaten.
- RECHENSTRUKTUREN UND FUNKTIONALE PROGRAMME greifen mit den Rechenstrukturen die algebraischen Strukturen auf. Auf den Rechenstrukturen lassen sich Terme und die Termersetzungssysteme bilden. Es werden u.a. die Terme der Aussagen- und Prädikatenlogik behandelt. Die Termersetzung bildet den Kern der funktionalen Programme.

1.1.3 Programmierkonzepte

Das algorithmische Denken und die Umsetzung von Algorithmen in lauffähige (Java-) Programme werden mit dem Ziel vermittelt, dass jeder Teilnehmer nach erfolgreicher Bearbeitung des Kursbuchs das Programmieren im Kleinen methodisch und praktisch beherrscht.

- IMPERATIVE PROGRAMMIERUNG vertieft die in den heute eingesetzten Programmiersprachen intensiv genutzten elementaren Sprachelemente und Datenstrukturen. Die in der imperativen Programmierung einsetzbaren Zusicherungen im Zusammenhang mit der bedingten Anweisung und den Schleifen (Schleifeninvariante) werden behandelt.

- OBJEKTORIENTIERTE PROGRAMMIERUNG fasst die Daten und darauf arbeitenden Funktionen als eine als Klasse bezeichnete Einheit auf und stellt eine Standardmethode zur strukturierten Programmierung zur Verfügung. Auf die wichtigsten dynamischen Datenstrukturen (Listen, Bäume und Graphen), die das Klassenkonzept nutzen, wird eingegangen.
- FORTGESCHRITTENE PROGRAMMIERKONZEPTE umfassen weitergehende Konzepte (z.B. Ausnahmebehandlung und Parallelität), die konzeptionell eingeführt werden und anhand konkreter Programmierbeispiele beschrieben werden.

Durch die in das Diagramm in Information 2 eingezeichneten Pfeile wird verdeutlicht, dass die Kursblöcke nicht sequentiell nacheinander bearbeitet werden. Der Grund hierfür ist, dass bereits zu einem möglichst frühen Zeitpunkt in der Vorlesung Programmierkonzepte soweit eingeführt werden sollen, dass das Programmieren im Kleinen eingeübt werden kann.

Die Inhalte der Kurseinheiten basieren auf etablierten deutschsprachigen Informatik-Lehrbüchern, wie in Information 3 näher ausgeführt ist.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

[Go97] Gerhard Goos: Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer-Verlag, 1997
Hauptquelle für die folgenden Kurseinheiten:
- EINFÜHRUNG IN DAS FACHGEBIET
- GRUNDBEGRIFFE DER INFORMATIK
- ALGEBRAISCHE STRUKTUREN UND FORMALE SYSTEME

[Br98] Manfred Broy: Informatik – Eine grundlegende Einführung, Band 1: Programmierung und Rechenstrukturen, Springer Verlag, 1998
- RECHENSTRUKTUREN UND FUNKTIONALE PROGRAMME
- IMPERATIVE PROGRAMMIERUNG

[Mö03] Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag, 2003
- PROGRAMMIERGRUNDLAGEN
- IMPERATIVE PROGRAMMIERUNG
- OBJEKTORIENTIERTE PROGRAMMIERUNG
- FORTGESCHRITTENE PROGRAMMIERKONZEPTE

Information 3: Der Veranstaltung zugrunde liegende und empfohlene Lehrbücher

Der Kursblock Formale Grundlagen orientiert sich an [Go97] und [Br98], während die Programmierkonzepte im Wesentlichen auf der Basis von [Mö03] aufgebaut sind. Die Kurseinheiten sind so konzipiert, dass die genannten Lehrbücher zur weiteren Vertiefung des Lernstoffes genutzt werden können. Aus diesem Grund sind die Kurseinheiten hinsichtlich der Struktur und der verwendeten Notation eng an die Lehrbücher angelehnt.

Neben diesen drei als Hauptquellen genutzten Lehrbüchern wurden noch einige weitere Quellen genutzt, die jeweils am Ende jeder Kurseinheit im Literaturverzeichnis aufgeführt sind.

1.2 Gestaltung eines Kursdokuments

Zu jeder behandelten Kurseinheit wird jeweils ein Kursdokument bereitgestellt, das die schriftliche Ausarbeitung (Skript) zu den in der Vorlesung vermittelten Inhalten darstellt. Das

vorliegende Kursdokument zur Kurseinheit mit dem Titel KURSBUCH-ÜBERBLICK UND ERGÄNZENDES INFORMATIKPORTAL ist ein konkretes Beispiel eines solchen Dokuments.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Titelseite
  - Titel
  - Kurzbeschreibung
  - Lernziele
  - Inhaltsverzeichnis
- Hauptteil
  - in Form von zwei oder mehr Kapiteln
  - Präsentationsmaterial als Grafiken (Information bzw. Interaktion) integriert
- Abschließender Verzeichnisteil
  - Glossar und Abkürzungen
  - Index
  - Informationen und Interaktionen
  - Literatur

Information 4: Aufbau eines Kursdokuments

Jedes Kursdokument ist gemäß den in Information 4 aufgeführten Teilen strukturiert. In den Kursdokumenten werden Schriftarten mit der folgenden Bedeutung durchgängig verwendet:

- Über-/Unterschriften und Hervorhebungen sind im Text **fett** markiert.
- Englischsprachige Begriffe sind *kursiv* geschrieben.
- Eigennamen (Unternehmen, Institutionen, Produkte, Personen, ...) und Code-Beispiele sind in ArialNarrow gestellt.

Eine zentrale Eigenschaft der Kurs- und Übungsdokumente besteht darin, dass das vom Dozenten verwendete Präsentationsmaterial als Grafiken in die Dokumente eingebunden ist. Die Materialien sind so gehalten, dass die Interaktion zwischen Dozent und Studierenden gefördert wird. So sind an verschiedenen Stellen Interaktionsfolien eingestreut, die von den Studierenden selbstständig zu bearbeitende Fragen und praktische Problemstellungen enthalten. Die Interaktionsfolien sind im Kursdokument mit der Bezeichnung Interaktion eingebunden, während die Folien ohne Interaktionsanteil als Information bezeichnet werden.

- Welche der in den nachfolgenden Kurseinheiten ausführlich behandelten Begriffe sind bereits (ungefähr) bekannt und was ist deren (grobe) Bedeutung?
  - Java _____
  - UML _____
  - Relation _____
  - Endlicher Automat _____
  - Rechenstruktur _____
  - Keller _____
  - Methode _____
  - Rekursion _____
  - Bedingte Anweisung _____

Interaktion 5: Die erste Interaktionsfolie

Interaktion 5 ist ein erstes Beispiel einer Interaktionsfolie, die optisch an dem Fragezeichen im oberen rechten Teil zu erkennen ist. In dieser Interaktion wird der Teilnehmer danach gefragt, inwieweit ihm gewisse zentrale Begriffe, die in den nachfolgenden Kurseinheiten behandelt werden, bereits bekannt sind. Lösungen zu dieser und allen weiteren Interaktionen der Vorlesung und der Übungen werden vom Dozenten in Zusammenarbeit mit den Teilnehmern innerhalb der Veranstaltung entwickelt.

Zu dem vorliegenden Kursbuch wird ein im nächsten Kapitel beschriebenes Portal angeboten, über das weitergehende Dienste und Lehrmaterialien – wie z.B. die Living Documents – bereitgestellt werden. Die Living Documents beinhalten u.a. die in der Veranstaltung durch den Dozenten gegebenen Lösungshinweise zu den oben beschriebenen Interaktionen.

2 INFORMATIK-I-PORTAL (IPO)

Ein (Web-) Portal ist gemäß [We04] eine *Web Site*, unter der eine Vielzahl an Web-Diensten verfügbar gemacht wird. Häufig haben Portale die Eigenschaft, die Dienste rollen- und aufgabenbezogen anzubieten, um hierdurch eine möglichst gezielte Unterstützung der Anwender zu erreichen.

Im Zusammenhang mit der Durchführung von Lehrveranstaltungen sind zahlreiche Dienste vorstellbar, die zur Unterstützung aller beteiligten Rollen und deren Aufgaben sinnvoll über ein Web-Portal angeboten werden können.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Den Studierenden angebotene Dienste
  - Bereitstellung aktueller organisatorischer Informationen
  - Zugriff auf die Aufgabenblätter
  - Teilnahme am Diskussionsforum (Registrierung erforderlich)
  - Bereitstellung weiterführender Materialien (Registrierung erforderlich)
    - Beispiele: Kursdokumente, LDocs (Living Documents) und Lösungsvorschläge zu den Übungsblättern

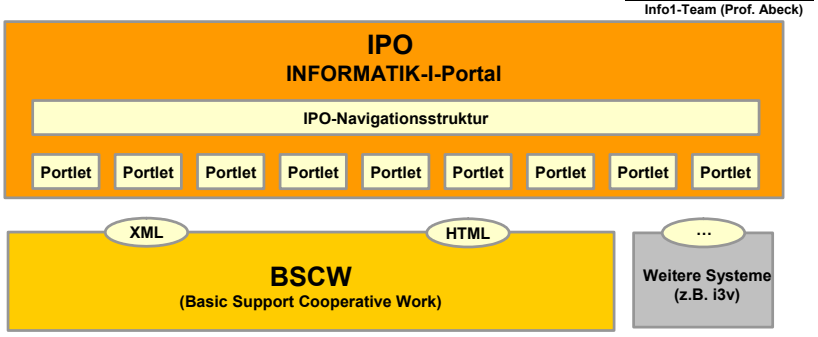
- Den Dozierenden zusätzlich angebotene Dienste
  - weitergehenden Kommunikations- und Kollaborationsunterstützung

Information 6: DAS INFORMATIK-I-PORTAL IPO - Angebotene Dienste

In Information 6 sind die Dienste aufgeführt, die durch das IPO – dem INFORMATIK-I-Portal – den Studierenden sowie den Dozenten angeboten werden. Das vorliegende Dokument konzentriert sich auf die IPO-Dienste für den Studierenden. Hinsichtlich einer näheren Beschreibung der Dienste, die von IPO den Dozenten (Tutoren, Übungsleiter, Vorlesungsdozent) bereitgestellt werden, sei auf [Sc04, Br04] verwiesen.

Eine zentrale Aufgabe des IPO ist aus der Sicht des Studierenden der gezielte Zugriff auf sämtliche organisatorischen und inhaltlichen Informationen zu der Lehrveranstaltung. Hierzu zählen u.a. die im vorhergehenden Kapitel eingeführten Kurseinheiten zur Vorlesung.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)



- Das IPO nutzt die Funktionalität verschiedener darunter liegender Systeme
  - wichtigstes System ist die Kollaborationsplattform BSCW
    - Ablage von Materialien, Diskussionsforum, aktuelle Informationen
  - andere bestehende Systeme unterstützen die Verwaltung von Lehrveranstaltungen oder die Einschreibung in Tutorien

- Die bestehenden Systeme werden unter Nutzung so genannter Portlets unter eine Web-Oberfläche eingebunden
  - als Portalsoftware wird Apache Jetspeed eingesetzt (Open Source)

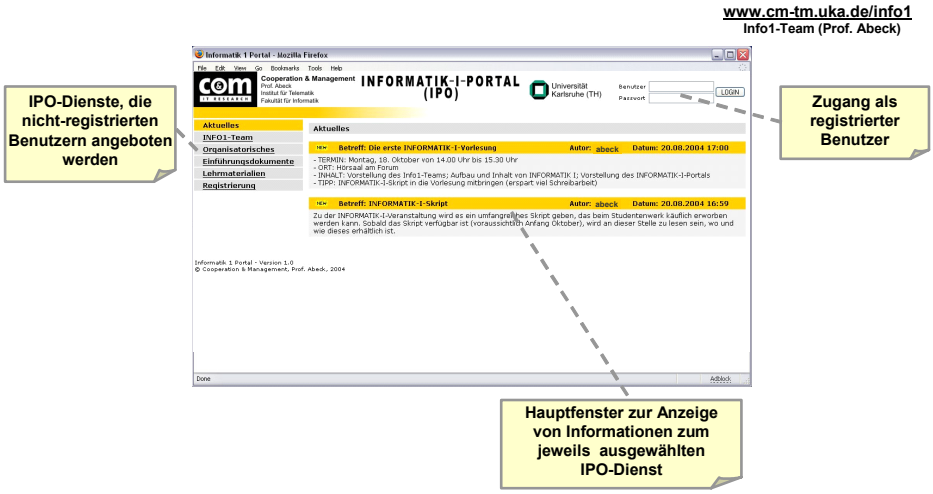
Information 7: Systemsicht auf das IPO

Vor einer detaillierten Beschreibung der Nutzung des IPO wird in Information 7 skizziert, welche Informatiksysteme sich dahinter verbergen. Das Portal selbst besteht aus einer

entsprechend angepassten und erweiterten *Open Source Software* (Apache Jetspeed), unter die verschiedene an der Fakultät für Informatik vorhandene Systeme eingebunden wurden. Ein für die von IPO bereitgestellte Funktionalität ist dabei eine Kollaborationsplattform (Basic Support Cooperative Work BSCW).

2.1 Zugang und Bedienung

Nach Aufruf eines Standard-Web-*Browsers* (z.B. Netscape, Mozilla, Internet Explorer) und der Eingabe der Web-Adresse http://www.cm-tm.uka.de/info1 erscheint die in Information 8 angegebene Oberfläche.



- Das IPO ist erreichbar über http://www.cm-tm.uka.de/info1

- Einem nicht-registrierten Benutzer stellt das IPO nur ein eingeschränktes Dienstangebot zur Verfügung

Information 8: Aufruf des IPO

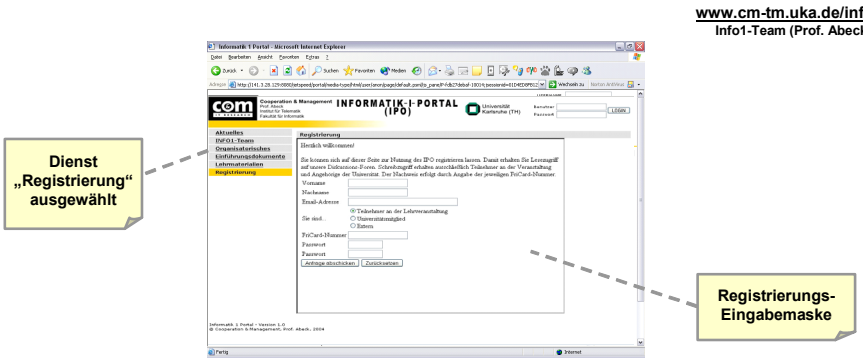
Wie die Übersicht im linken Fenster zeigt, stehen einem IPO-Benutzer, der noch nicht registriert ist, nur eingeschränkte Dienste zur Verfügung. Diese Dienste sind:

- Aktuelles: Hierunter können alle Ankündigungen des Vorlesungsdozenten oder Übungsleiters abgerufen werden. Beim Aufruf des IPO werden diese Informationen im Hauptfenster des Portals (siehe Information 8) angezeigt.
- Info1-Team: Alle Personen, die dazu beitragen, dass die INFORMATIK-I-Veranstaltung in der gegebenen Form durchgeführt werden kann, sind hierunter aufgeführt.
- Organisatorisches: Informationen, die im Verwaltungssystem der Fakultät für Informatik zu der Veranstaltung INFORMATIK I verfügbar sind.
- Einführungsdokumente: Neben dem vorliegenden Dokument KURSBUCH-ÜBERBLICK UND ERGÄNZENDES NFORMATIKPORTAL steht außerdem ein Dokument, das den Ablauf der Veranstaltung beschreibt, zum *Download* bereit.
- Lehrmaterialien:
    - Übungsblätter: Unter diesem Punkt können die Übungsblätter zur Veranstaltung herunter geladen werden.
    - Software: Hierüber wird die zur Lösung der Übungsblätter benötigte Software bereitgestellt.
- Registrierung: Durch Anklicken dieses Dienstes ist eine im nächsten Abschnitt näher beschriebene Registrierung beim IPO möglich, um weitergehende Dienste (u.a. Zugriff auf alle Materialien wie beispielsweise die weiter unten näher beschriebenen Living Documents,

Lösungsvorschläge zu den Übungsblättern oder die Teilnahme am Diskussionsforum) in Anspruch nehmen zu können.

2.2 Registrierung

Mit der Registrierung wird ermöglicht, dass der Anbieter der IPO-Dienste bei Bedarf mit den IPO-Dienstnehmern in Kontakt treten kann, weshalb die Angabe der gültigen *E-Mail*-Adresse eine zentrale Information darstellt, die bei der Registrierung vom IPO-Anwender abgefragt wird.
.



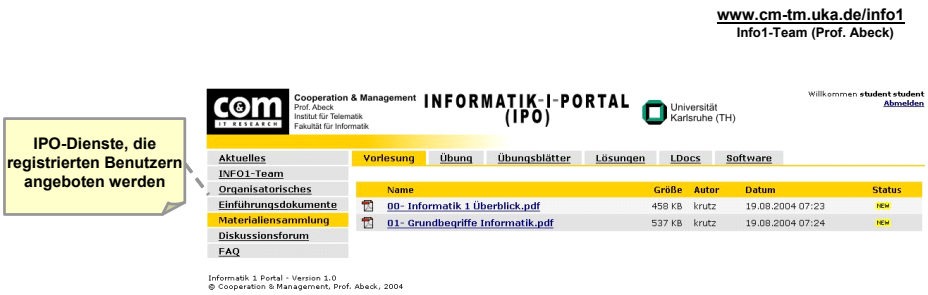
- Gültige E-Mail-Adresse und vollständiger Vor- und Nachname
  - keine Spitznamen

- Unterscheidung zwischen Benutzern innerhalb und außerhalb der Universität
  - eigene Beiträge zum Diskussionsforum wird ausschließlich den Universitätsangehörigen ermöglicht

Information 9: Registrierung beim IPO

Die in Information 9 gezeigte Registrierungsmaske erscheint nach Auswahl des IPO-Dienstes Registrierung (siehe linkes Fenster). Neben der erwähnten (gültigen) *E-Mail*-Adresse sind der (korrekte) Vorname und Nachname anzugeben.
Im Rahmen der Registrierung wird zwischen Angehörigen der Universität Karlsruhe (TH) und IPO-Benutzern außerhalb der Universität unterschieden. Universitätsangehörige sind aufgefordert, die Nummer ihrer FriCard anzugeben. Hierdurch erlangen diese Benutzer die Berechtigung, eigene Beiträge im Diskussionsforum bereitzustellen.

Nachdem das Registrierungsformular ausgefüllt und durch Anklicken des Absenden-Feldes abgeschickt wurde, erhält der Benutzer kurzfristig eine (an die im Registrierungsformular angegebene *E-Mail*-Adresse gesendete) Antwort-*Mail*, in der alle Informationen enthalten sind, die für eine erfolgreiche Anmeldung als registrierter Benutzer erforderlich sind.



- Zugriff auf alle bereitgestellten Informationen zur Vorlesung und zur Übung
  - u.a. auf die Veranstaltungsaufzeichnungen im Form von Living Documents

- Teilnahme am Diskussionsforum
  - aktiv (Universitätsangehörige) oder passiv (externe Gäste)

Information 10: IPO-Dienstangebot für registrierte Benutzer

Information 10 zeigt im linken Fenster die einem registrierten Benutzer zur Verfügung stehende Liste der IPO-Dienste. Die zu den bereits beschriebenen Diensten neu hinzugekommenen Dienste sind:

- Materialiensammlung: Über diesen Punkt können alle Dokumente zur INFORMATIK-I-Veranstaltung heruntergeladen werden. Die Dokumente sind die in folgenden Kategorien unterteilt:
  - Vorlesung: Hierüber werden alle Kursdokumente, die in der Vorlesung behandelt werden, vollständig bereitgestellt.
  - Übung: Entsprechende Kursdokumente, die in der Zentralübung besprochen werden.
  - Lösungen: Lösungsvorschläge zu den bisher behandelten Übungsblättern.
  - LDocs: Unter dieser Kategorie werden die als Living Documents bezeichneten multimedialen Dokumente sowohl zum *Online*-Abruf als auch zum *Download* zur *Offline*-Nutzung bereitgestellt (siehe Beschreibung im nächsten Kapitel).
- Diskussionsforum: Hierüber gelangt man zu dem im nächsten Abschnitt näher beschriebenen Diskussionsforum.
- FAQ: Über diesen Punkt wird eine Liste von Antworten zu häufig gestellten Fragen bereitgestellt.

2.3 Diskussionsforum

Das Diskussionsforum kann und soll von registrierten Benutzern, die Universitätsangehörige sind (hierzu gehören insbesondere die Teilnehmer der INFORMATIK-I-Veranstaltung) zum aktiven Austausch von Informationen, Gedanken, Ideen, Fragen und Antworten intensiv genutzt werden. Dabei müssen einige Verhaltensregeln beachtet werden. Wenn sich alle an diese Regeln halten, steht einer interessanten Diskussion und einem lebhaften Meinungsaustausch nichts im Wege.

Keinesfalls geduldet werden rassistische oder strafrechtlich relevante Beiträge jeglicher Art. Darunter fallen beispielsweise Beiträge, die NS-Propaganda in die Foren tragen, aber auch Hinweise auf Web-Seiten mit pornographischem Inhalt und URLs zu menschenverachtenden Seiten. Wer hier öffentlich zu Straftaten aufruft oder in strafrechtlich relevanter Weise droht,

muss ebenfalls mit entsprechenden rechtlichen Konsequenzen rechnen. Auch das Raubkopieren (z.B. mp3, Filme) ist kein Kavaliersdelikt.

Sollten Sie auf derartige Beiträge stoßen, informieren Sie bitte das Info1-Team (z.B. durch eine *Mail* an info1@cm-tm.uka.de).

- Das IPO-Forum dient zum aktiven Austausch von Informationen, Gedanken, Ideen, Fragen und zugehörigen Antworten
  - grundsätzlich gilt: rassistische Inhalte jeder Art und strafrechtlich relevante Inhalte werden nicht geduldet
- Bei jeder Kommunikation sollten gewisse Verhaltensregeln eingehalten werden
  - diese Regeln werden für das Forum durch die IPO-Netiquette vorgegeben
- Ausschnitt aus der IPO-Netiquette
  - Seien Sie immer freundlich, höflich und hilfsbereit
  - Atmen Sie erstmal tief durch, wenn man Sie angreift
  - Seien Sie tolerant gegenüber den Meinungen anderer
- Die IPO-Netiquette sollte von jedem Forumsteilnehmer eingehalten werden

Information 11: Die IPO-Netiquette

Im Folgenden sind die wichtigsten Verhaltensregeln in Form einer IPO-Netiquette (siehe Information 11) aufgeführt, die bitte bei der Nutzung des IPO-Forums zu beachten sind:

- Seien Sie immer freundlich, höflich und hilfsbereit.
- Seien Sie tolerant gegenüber den Meinungen anderer.
- Atmen Sie erstmal tief durch, wenn man Sie angreift. Antworten Sie darauf erst nach einer "Abkühlpause", und lassen Sie sich nicht auf das Niveau des Angreifers herabziehen.
- Schreiben Sie möglichst kurz und prägnant.
- Geben Sie bei Zitaten und Referenzen die Quelle an.
- Lesen Sie sich Ihren Beitrag vor dessen Einstellen in das Forum noch einmal in Ruhe durch und achten Sie auf eine angemessene Ausdrucksweise.
- Geben Sie im Zweifelsfall keine Kommentare ab. Vermeiden sie also Mitteilungen, die eine persönliche Beleidigung enthalten und weitere Beschimpfungen nach sich ziehen.
- Missbrauchen Sie das Diskussionsforum nicht als *E-Mail*-Ersatz, um eine einzelne Person anzusprechen.
- Behandeln Sie andere Forumsteilnehmer so, wie Sie von Ihnen behandelt werden möchten.
- Gehen Sie nicht davon aus, dass Sie immer so verstanden werden, wie Sie es gemeint haben.
- Helfen und unterstützen Sie Kommilitonen höflich, auch wenn Ihnen die Fragen sehr einfach oder trivial erscheinen. Vermeiden Sie Formulierungen wie "ist ganz easy"; damit helfen sie dem Fragenden nicht weiter, sondern demotivieren ihn nur.
- Konzentrieren Sie sich auf ein Thema pro Nachricht und fügen Sie eine aussagekräftige Betreffzeile (Subject) hinzu, so dass Nachrichten leicht gefunden werden können. Die Betreffzeile entscheidet darüber, ob Ihr Beitrag gelesen wird oder nicht.
- Vermeiden Sie Betreffzeilen wie Wichtige Frage !!! oder Brauche ganz dringend Hilfe. Der größte Teil der Forumsteilnehmer wendet sich an das Forum, um Fragen zu klären.
- Wenn Sie andere Beiträge zitieren, so löschen Sie alles, was nicht zum Verständnis der Antwort notwendig ist.

- Verwenden Sie Abkürzungen nur dann, wenn Sie sicher sein können, dass sich auch verstanden werden. Vorsicht: Zu viele Abkürzungen machen einen Beitrag unleserlich.
- Um den Umfang des Forums nicht zu sprengen, sollte ein offensichtlicher Zusammenhang der Beiträge mit der Veranstaltung vorhanden sein.

Was wir uns als Umgangsformen in den IPO-Diskussionsforen wünschen, ist übrigens nicht neu. Vergleichbare Verhaltensregeln gibt es auch in anderen *Newsgroups*.

3 LIVING DOCUMENTS (LDOCS)

Mit den LDocs wurde in der Forschungsgruppe Cooperation & Management (C&M) eine neue Form des Multimedia-Einsatzes in der Lehre entwickelt. Das Konzept wurde bereits in verschiedenen anderen Veranstaltungen (Vertiefungsfach-Vorlesung INTERNET-SYSTEME UND WEB-APPLIKATIONEN [C&M-ISWA] oder Wahlpflichtfach-Veranstaltung KOMMUNIKATION UND DATENHALTUNG [C&M-KuD]) erfolgreich erprobt. Da die Rückmeldungen der Studierenden sehr positiv waren [FI04], werden die LDocs auch für alle Vorlesungs- und Übungsinhalte der INFORMATIK-I-Veranstaltung über das IPO zur Verfügung gestellt.

Bevor auf die technischen Details eingegangen, durch die die LDocs (sowohl *online* als auch *offline*) genutzt werden können, werden zunächst die mit diesem Konzept verfolgten inhaltlichen Ziele beschrieben.

3.1 C&M-konformer Kurs als Ausgangspunkt

Ein erheblicher Anteil des für die Bereitstellung eines Lehrangebots erforderlichen Zeitaufwands wird durch die Erstellung und ständige Aktualisierung des zumeist in Form von Folien vorliegenden Präsentationsmaterials verursacht. Durch die Arbeit an diesem Material setzt sich der Lehrende aktiv mit den Inhalten auseinander, die er im Rahmen einer Lehrveranstaltung (z.B. Seminar, Vorlesung, Übung, Praktikum) dem Lernenden näher bringt. Dem Lernenden dient (zumindest ein Teil) des vom Dozenten genutzten Präsentationsmaterials als wichtige Grundlage zur Vor- und Nachbereitung des Lehrinhalts.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Beobachtung: Das vom Lehrenden als Präsentationsmaterial in einer Lehrveranstaltung genutzte Lehrmaterial (häufig: Foliensammlung) ist als Lernmaterial nicht geeignet
- Idee: Konsequente Entwicklung eines anspruchsvollen Lernmaterials aus dem Präsentationsmaterial
- Lösung: C&M-konformer Kurs
  - erfüllt die Anforderungen, die an ein gutes Lehrbuch gestellt werden (ausformulierter Text, Index, Abkürzungs- und Literaturverzeichnisverzeichnis, ...)
  - beinhaltet vollständig (das in Form von Folien) vorliegende Präsentationsmaterial gemäß bestimmter einzuhaltender Konventionen
  - Interaktionsfolien als wichtiges pädagogisches Element, das den Lernenden zur aktiven Mitarbeit in der Lehrveranstaltung animiert

Information 12: Motivation des C&M-konformen Kurses

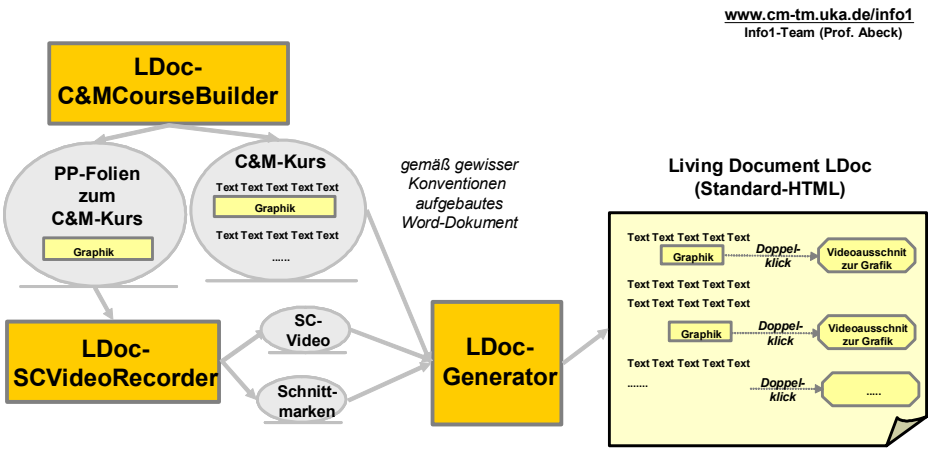
Die Praxis macht deutlich, dass das vom Lehrenden genutzte Präsentationsmaterial häufig nicht als Lernmaterial für den Lernenden geeignet ist. So kann z.B. eine (noch so umfangreiche) Foliensammlung ein gut strukturiertes und klar geschriebenes Lehrbuch nicht ersetzen. Wünschenswert wäre aus der Sicht eines Lernenden ein Lehrbuch, das das in der Lehrveranstaltung genutzte Präsentationsmaterial zu einem für die Vor- und Nachbereitung des Stoffs geeigneten Lehrbuchs erweitert.

Der Ansatz des C&M-konformen Kurses stellt ein pragmatisches und einfach anzuwendendes Konzept zur konsequenten Verknüpfung von Präsentations- und Lernmaterial auf der Basis der allgemein genutzten MS-Office-Werkzeuge dar. Sämtliches für die INFORMATIK-I-Veranstaltung enteickelte Lehr- und Lernmaterial – und damit auch das vorliegende Dokument – wurde gemäß diesem Konzept erstellt.

3.2 Vom C&M-konformen Kurs zu den LDocs

Die Tatsache, dass eine enge Kopplung zwischen Lehr- und Lernmaterial erreicht wurde und bei der Kurserstellung einige einfache Konventionen eingehalten wurden, eröffnet eine neue Möglichkeit hinsichtlich der Erweiterung dieser Materialien um multimediale Elemente. So zeichnet C&M bereits seit Jahren seine Veranstaltungen als *Screen Capture Videos* auf, die die Stimme des Dozenten und den Bildschirminhalt (engl. *Screen*) "einfängt" (engl. *Capture*). Durch den Einsatz eines eingabesensitiven Bildschirms in Form eines Grafiktabletts oder eines Plasmabildschirms lassen sich auf diese Weise die während der Veranstaltung vom Lehrenden entwickelten elektronischen Tafelbilder (inklusive der dazu gemachten Erklärungen) aufzeichnen.

Die mit diesem *Screen Capture Videos* gesammelten Erfahrungen verdeutlicen, dass diese zwar von den Lernenden gut angenommen werden. Allerdings haben die Videos nicht nur Vorteile: Jeder, der bereits mit einem solches Video gearbeitet hat, stellt fest, dass es schwer fällt, über längere Zeit den Inhalten aufmerksam zu folgen. Aus dieser Beobachtung resultierte das Ziel, die Videos stärker mit den bestehenden Lernmaterialien, in diesem Fall dem C&M-konformen Kurs, zu koppeln. Die Idee führt unmittelbar zum Living Document, kurz LDoc: Ein Lernender, der sich den Lehrstoff erarbeitet, hat die Möglichkeit, das im C&M-konformen Kurs enthaltene Präsentationsmaterial gewissermaßen "zum Leben zu erwecken". Durch einfaches Anklicken einer der eingebundenen Folien wird der entsprechende Teil des *Screen Capture Videos* angezeigt, auf dem der Lehrende diese Folie erklärt und ggf. annotiert oder um ein elektronisches Tafelbild ergänzt.



- LDoc-C&MCourseBuilder: auf Office-Werkzeugen basierende Erstellung eines C&M-konformen Kurses
- LDoc-SCVideoRecorder: Aufzeichnung und Schnittmarken-Erzeugung
- LDocGenerator: Zuordnung der Video-Ausschnitte zu den Grafiken und Bereitstellung als HTML-Dokument

Information 13: Erzeugung eines LDoc aus einem C&M-konformen Kurs

Die Abbildung in Information 13 zeigt auf der rechten Seite den Aufbau eines LDoc, bestehend aus Text und Abbildungen sowie den ergänzten dynamischen Elementen, die neben den *Screen Capture Videos* auch andere multimediale Elemente, wie z.B. Annotationen sein können.
C&M hat eine Werkzeuglösung LDoc-Tools (LivingDocument-Tools) bestehend aus den drei in Information 13 angegebenen Werkzeugen entwickelt, durch die die Erzeugung eines *Living Document* aus einem C&M-konformen Kurs und einer als *Screen Capture Video* vorliegenden Aufzeichnung der Lehrveranstaltung gezielt unterstützt wird.

3.3 INFORMATIK-I-LDocs

Der einfachste Weg, sich ein eigenes Bild von den LDocs zu machen, besteht darin, diese im Rahmen der Beschäftigung mit dem INFORMATIK-I-Lerninhalt einzusetzen.

3.3.1 Erforderliche Software

Die LDocs bestehen aus einem HTML-Dokument, in dem die Videos (avi-Format) über *Hyperlinks* eingebunden sind. Zum Darstellen des Dokuments und Abspielen des Videos werden daher ein *Browser* sowie ein *Media Player* benötigt.

- LDocs sind HTML-Dokumente mit eingebundenen Videos
  - als Software wird daher ein Standard-Web-Browser und ein Media Player benötigt

- Windows-Umgebung
  - beliebiger Browser
  - Windows Media Player
  - spezieller Video-Codec (TechSmith)
    - steht kostenfrei zur Verfügung unter http://www.techsmith.com/products/studio/codecdownload.asp

- Linux-Umgebung
  - beliebiger Browser
  - Media Player mplayer und zugehöriges essential codecs package
    - stehen kostenfrei zur Verfügung unter http://www.mplayerhq.hu

Information 14: Zur LDoc-Nutzung erforderliche Software

Es stehen für die Nutzung der LDocs sowohl unter Windows als auch unter Linux geeignete Softwareprodukte zur Verfügung (siehe Information 14).

Windows

Bei einer Standard-Installation eines aktuellen Windows-Betriebssystems (Windows 2000 oder Windows XP) werden mit dem Internet Explorer und dem Windows Media Player bereits zwei zum Nutzen der LDocs verwendbare Programme mitgeliefert. Es sind aber natürlich auch andere *Browser* wie Mozilla (http://www.mozilla.org) und andere Video-Abspielprogramme, z.B. Media Player Classic (http://sourceforge.net/projects/guliverkli/) einsetzbar.
Da zur Kodierung des Videos ein speziell für *Screen Capture Videos* geeigneter *Video-Codec* von TechSmith verwendet wird, muss dieser zum Abspielen der Videos installiert werden. Der benötigte *Video-Codec* steht im Web kostenfrei zur Verfügung (siehe http://www.techsmith.com/products/studio/codecdownload.asp). Nach Herunterladen des *Video-Codec* auf den eigenen Rechner kann dieser durch Doppelklick auf die Datei und einer anschließenden Bestätigung installiert werden. Damit sind alle Anforderungen an die Nutzung der LDocs unter Windows erfüllt.

Linux

Nach der Standard-Installation einer aktuellen Linux-Distribution stehen eine graphische Oberfläche und ein Standard-*Browser* zur Verfügung.
Unter Linux wird zum Abspielen der LDocs neben dem *Video-Codec* noch ein spezielles Video-Abspielprogramm benötigt. Dieses Programm mit dem Namen mplayer und das zugehörige essential codecs package kann unter http://www.mplayerhq.hu. herunter geladen werden. Nach erfolgreicher Installation kann mit den LDocs unter Linux gearbeitet werden.

3.3.2 Zugriff und Nutzung

Die LDocs werden von der Forschungsgruppe Cooperation & Management über das Internet in zwei Versionen angeboten:
1) *Online* als Verweis auf eine HTML-Datei
2) *Offline* als zip-Datei zum Herunterladen und lokaler Nutzung

- Zugriff auf die LDocs
  - Aufruf des INFORMATIK-I-Portals (www.cm-tm.uka.de/info1)
  - Anmelden als registrierter Benutzer
  - die LDocs befinden sich unter dem IPO-Dienst „Materialiensammlung" in der Kategorie „LDocs"

- Online-Nutzung
  - Auswahl eines HTML-Links und Online-Nutzung des LDocs

- Offline-Nutzung
  - Download des gewünschten LDocs
  - Entpacken der zip-Datei

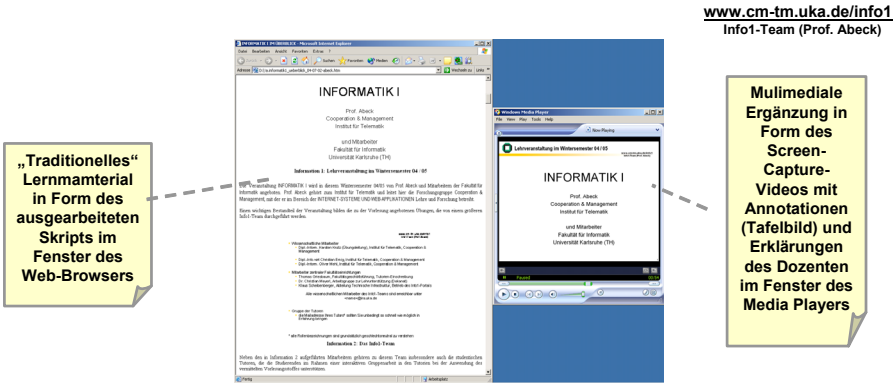
Information 15: Zugriff und Nutzung der INFORMATIK-I-LDocs

Sowohl die *Online*- als auch die *Offline*-Version jedes LDocs befindet sich im INFORMATIK-I-Portal (IPO) unter dem IPO-Dienst Materialiensammlung in der Kategorie LDocs (siehe Information 15). Der Zugriff auf den Bereich im IPO, in dem die LDocs liegen, ist erst nach der im vorhergehenden Kapitel beschriebenen Registrierung am Portal möglich.

Der Vorteil der *Online*-Nutzung besteht darin, dass die LDocs sofort aufgerufen werden können. Der Nachteil ist, dass während der gesamten Zeit, in der mit den LDocs gearbeitet wird, eine Internet-Verbindung bestehen muss.
Falls eine *Online*-Nutzung nicht in Frage kommt, bietet sich die *Offline*-Nutzung an. Durch Anklicken des gewünschten Dokuments wird der *Download* auf den eigenen Rechner angestoßen und die komprimierte Datei (zip-Format) in dem angegebenen Verzeichnis abgelegt. Danach muss die Datei noch mit einem gängigen Dekomprimierungsprogramm entpackt werden. Der Aufruf eines LDoc erfolgt durch Öffnen der HTML-Datei im Standard-*Browser*.

3.3.3 Einstellungen des Web-Browsers und des Media Players

Abschließend werden einige Empfehlungen zu Einstellungen der beiden Programme gegeben, über die die LDocs genutzt werden.



- Die Oberfläche kann so eingestellt werden, dass beide Formen des Lernmaterials verzahnt genutzt werden können

- Einstellungen betreffen
  - Media Player als eigenes Fenster außerhalb des Web-Browsers ablaufen lassen
  - das gewünschte Erscheinungsbild des Media-Player-Fensters

Information 16: Empfohlene Einstellung der Oberfläche

Eine mögliche Anordnung des *Browser*-Fensters und des *Media-Player*-Fensters zeigt Information 16. Durch die Anordnung der beiden Fenster wird erreicht, dass der LDoc-Benutzer gemäß der „traditionellen“ Wissensaneignung das ausgearbeitete Kursdokument lesen kann und ergänzend bzw. parallel das *Screen-Capture*-Video mit den Annotationen und den Erklärungen des Dozenten verfolgen kann.
Im Falle der Nutzung des Internet Explorer wird der Benutzer beim ersten Anklicken einer Graphik im LDoc danach gefragt, of die Videos im Internet Explorer oder extern ablaufen sollen. Es empfiehlt sich, hierbei aufgrund besserer Einstellungsmöglichkeiten die externe Variante zu wählen. Hierdurch ergibt sich dann die Möglichkeit, das Text/Grafik-Fenster des Web-Browsers und das Video-Fenster des *Media-Players* geeignet zu positionieren.
Wird beispielsweise der Microsoft Media Player zum Abspielen der Videos genutzt, wird empfohlen, die zahlreichen Kontrollelemente, die um das eigentlich relevante Video-Fenster angeordnet sind, durch entsprechende Einstellungen im Menüpunkt Ansicht auszublenden, so dass ein Erscheinungsbild des *Media Players* entsteht, wie in Information 16 angegeben ist.

Empfehlungen zur Nutzung der LDocs

Die Teilnehmer der INFORMATIK-I-Veranstaltung sollten das Portal und die LDocs als eine Ergänzung zum Besuch der Vorlesung, der Übung und der Tutorübung ansehen. Das Ziel der bereitgestellten rechnergestützten Lernsysteme ist eine effiziente Vor- und Nachbereitung des Lernstoffs. Das persönliche Gespräch mit den Dozenten und den Tutoren sowie das kollaborative Lösen von Problemen in einer Lerngruppe sollen und können hierdurch nicht ersetzt werden.

Danksagung

Das in diesem Dokument beschriebene INFORMATIK-I-Portal und die Living Documents sind aus Arbeiten in der Forschungsgruppe Cooperation & Management (C&M) hervorgegangen, an denen viele Personen mitgewirkt haben. Insbesondere haben zahlreiche Studierende wertvolle Beiträge im Rahmen der in der Forschungsgruppe durchgeführten Praktika sowie Studien- und Diplomarbeiten geleistet.

VERZEICHNISSE

Abkürzungen und Glossar

Abkürzung oder Begriff	Langbezeichnung und/oder Begriffserklärung
ATIS	Abteilung Technische Infrastruktur Name des internen Dienstleisters der Fakultät für Informatik der Universität Karlsruhe (TH).
BSCW	Basic Support Cooperative Work System zur Bereitstellung und Verwaltung von Inhalten über das Web.
C&M	Cooperation & Management Name der zum Institut für Telematik der Universität Karlsruhe (TH) gehörenden Forschungsgruppe.
FriCard	Fridericiana Card Bezeichnung der in der Universität Karlsruhe (TH) eingesetzten Ausweiskarte zur Regelung des Zugangs zu zahlreichen Universitätsressourcen (z.B. Räume, Bibliothek, Mensa).
IPO	INFORMATIK-I-Portal Bezeichnung des Web-Portals, das zu der im Wintersemester 04/05 an der Universität Karlsruhe (TH) stattfindenden Veranstaltung INFORMATIK I angeboten wird.
IPO-Netiquette	Verhaltensregeln, die bei der aktiven Nutzung des im IPO angebotenen Diskussionsforums zu beachten sind.
Kursblock	Teil einer Lehrveranstaltung, in dem eine überschaubare Anzahl von inhaltlich zusammengehörigen Kurseinheiten enthalten ist.
Kurseinheit	Inhalt zu einem abgeschlossenen Thema mit einem Umfang von ca. 2 bis 4 Vorlesungsveranstaltungen (90 Minuten). Zu jeder Kurseinheit besteht ein einheitlich aufgebautes Kursdokument. Vorliegendes Dokument ist ein Beispiel eines Kursdokuments zur Kurseinheit KURSBUCH-ÜBERBLICK UND ERGÄNZENDES INFORMATIKPORTAL.
LDoc	Living Document Ein in Standard-HTML-Format vorliegendes Textdokument (Document) mit eingebundenen Grafiken, die durch Anklicken um (i.d.R. multimediales) Material ergänzt werden können und damit zum Leben (Living) erweckt werden.
LDocGenerator	Living Document Generator Ein von C&M entwickeltes Werkzeug zur Erstellung der LDocs.
Screen-Capture-Video	Aufzeichnung des Bildschirminhalts (*Screen*) und des gesprochenen Worts; besonders geeignet bei Verwendung eines eingabesensitiven Bildschirms (Grafiktablett, Plasmabildschirm).

TSCC — Techsmith Screen Capture Codec
Video-Codec, der zur verlustfreien Komprimierung von Videodateien (insbesondere von *Screen-Capture*-Aufnahmen) verwendet wird.

(Web-) Portal — *Web Site*, unter der eine Vielzahl an Web-Diensten verfügbar gemacht wird.

Index

Informationen und Interaktionen

Literatur

[AB+03] Sebastian Abeck, Pascal Bihler, Karsten Krutz, Christian Mayerl, Mads Stavang, Marco Willsch, C&M-konformer Kurs und Living Document, GI-Jahrestagung, Ulm, 2003.

[AG+04] Sebastian Abeck, Markus Gebhard, Karsten Krutz, Klaus Scheibenberger, Niko Schmid: Ein Portal zur Lehrunterstützung, Arbeitskonferenz "Elektronische Geschäftsprozesse", Klagenfurt, 2004.

[Br04] Frank Brandt, Anbindung eines Content-Management-Systems an ein Portal, Studienarbeit, Universität Karlsruhe (TH), C&M (Prof. Abeck), 2004.

[Br98] Manfred Broy: Informatik – Eine grundlegende Einführung, Band 1: Programmierung und Rechenstrukturen, Springer Verlag 1998.

[C&M-ASA] Cooperation&Management, ALLGEMEINE STRUKTURIERUNGS- UND ARCHITEKTURPRINZIPIEN, Kursdokument zur Vorlesung "Kommunikation & Datenhaltung", http://www.cm-tm.uka.de/kud,

Universität Karlsruhe (TH), C&M (Prof. Abeck)

[C&M-ISWA] Cooperation&Management, Kursdokumente zur Vorlesung "INTERNET-SYSTEME UND WEB-APPLIKATIONEN (ISWA)", http://www.cm-tm.uka.de/iswa, Universität Karlsruhe (TH), C&M (Prof. Abeck).

[C&M-KuD] Cooperation&Management, Kursdokumente zur Vorlesung "KOMMUNIKATION UND DATENHALTUNG (K&D)", http://www.cm-tm.uka.de/iswa, Universität Karlsruhe (TH), C&M (Prof. Abeck).

[Go97] Gerhard Goos: Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer Verlag 1997.

[Mö03] Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

[Sc04] Niko Schmid, Entwicklung eines Portals zur Unterstützung ausgewählter Geschäftsanwendungsfälle in der Aus- und Weiterbildung, Diplomarbeit, Universität Karlsruhe (TH), C&M (Prof. Abeck), 2004.

[We04] Webopedia, Web portal, Online encyclopedia, http://www.webopedia.com/TERM/W/Web_portal.html, 2004.

EINFÜHRUNG IN DAS FACHGEBIET

Kurzbeschreibung

Es wird ein Überblick über das Fachgebiet der Informatik mit dem Informatiksystem als dessen zentralen Untersuchungsgegenstand gegeben und das Tätigkeitsprofil eines Informatikers wird skizziert.

Schlüsselwörter

Informatik-Teilgebiete, Informatiksystem, System, Systemkonstruktion, Modellierung, Informatiker-Tätigkeitsprofil, Kompetenzen, Beispielszenarien

Lernziele

1. Die wesentlichen Teilgebiete der Informatik sind bekannt.
2. Informatiksysteme und deren verantwortungsvolle Nutzung werden als die zentrale Fragestellung der Informatik begriffen.
3. Das Tätigkeitsprofil eines Informatikers in der beruflichen Praxis kann nachvollzogen werden.

Hauptquellen

- Fakultät für Informatik der Universität Karlsruhe (TH): Fakultäts-Web, www.ira.uka.de
- Gerhard Goos: Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer Verlag 1997.
- Joerg E. Staufenbiel, Birgit Giesen (Hrsg.): Berufsplanung für den IT-Nachwuchs, Staufenbiel Institut für Studien- und Berufsplanung, 2001.

Inhaltsverzeichnis

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- TEILGEBIETE DER INFORMATIK
  - Klassische Einteilung, Strukturierung gemäß des Informatik-Studiengangs
- INFORMATIKSYSTEME
  - Systemeigenschaften, Systemtypen, Modellierung eines Systems, Systembeispiel
- TÄTIGKEITSPROFIL EINES INFORMATIKERS
  - Berufsfelder, Tätigkeitsschwerpunkte von Hochschulabsolventen

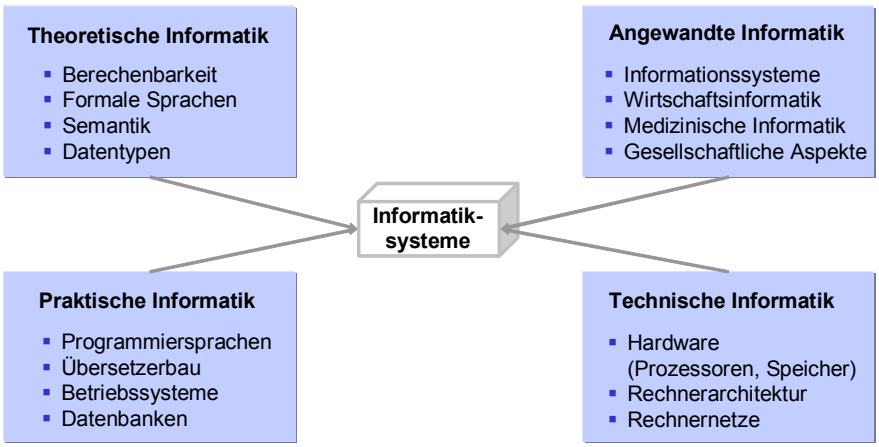
Information 1: ÜBERBLICK ÜBER DAS FACHGEBIET

1 TEILGEBIETE DER INFORMATIK

Die Informatik ist noch eine relativ junge Wissenschaft. Sie hat sich in den letzten Jahrzehnten so rasant entwickelt, dass sie heute eine zentrale Rolle in praktisch allen Bereichen unserer Informationsgesellschaft einnimmt.

Die inhaltliche Gliederung des Fachgebiets der Informatik verdeutlicht, dass das Informatiksystem der zentralen Untersuchungsgegenstand ist (siehe Abbildung in Information 2).

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)



- Jedes Informatik-Teilgebiet trägt ganz gewisse Aspekte zur Entwicklung und zum Einsatz der Informatiksysteme bei

Information 2: ÜBERBLICK ÜBER DAS FACHGEBIET - Teilgebiete der Informatik

Entsprechend der unterschiedlichen Beiträge zur Entwicklung eines Informatiksystems lässt sich die Informatik in die vier Teilgebiete der theoretischen, praktischen, technischen und angewandten Informatik einteilen. In Information 2 sind die wichtigsten Themen, die in diesen Teilgebieten behandelt werden, aufgelistet.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Theoretische Informatik
  - Theoretische Analyse und Konzeption von Informatiksystemen
- Praktische Informatik
  - Organisatorische und softwaretechnische Gestaltung von Informatiksystemen
- Technische Informatik
  - Aufbau von Informatiksystemen mit Schwerpunkt auf den technischen Komponenten
- Angewandte Informatik
  - verantwortungsvoller Einsatz von Informatiksystemen in beliebigen Anwendungsfeldern

Information 3: Beiträge der Teilgebiete zum Informatiksystem

Die Beiträge jedes der Teilgebiete zum Informatiksystem beschreibt Information 3.

Es bestehen auch verschiedene hiervon geringfügig abweichende Einteilungen. So wird z.B. in [BH03] neben der theoretischen und der technischen Informatik das Teilgebiet der systembezogenen Informatik unterschieden.

Eine feinere inhaltliche Gliederung der Informatik liefert der von der Fakultät für Informatik an der Universität Karlsruhe (TH) angebotene Diplom-Studiengang, und hier speziell die im Hauptstudium angebotenen so genannten acht Wahlpflichtfächer.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Die im Hauptstudium angebotenen acht Wahlpflichtfächer stellen ebenfalls eine Gliederung des Fachgebiets dar:

Welchen Teilgebieten (Theoretische / Praktische / Technische / Angewandte Informatik) lässt sich das Wahlpflichtfach schwerpunktmäßig zuordnen?

1. Formale Systeme _____
2. Algorithmentechnik _____
3. Softwaretechnik _____
4. Systemarchitektur _____
5. Kommunikation & Datenhaltung _____
6. Rechnerstrukturen _____
7. Echtzeitsysteme _____
8. Kognitive Systeme _____

Interaktion 4: Eine andere Form der inhaltlichen Gliederung des Fachgebiets

Nachfolgend sind die in den Wahlpflichtfächern behandelten Inhalte kurz beschrieben. Aufgrund dieser Beschreibung soll in Interaktion 4 [FI04] eine Zuordnung zu einem der vier oben eingeführten Teilgebiete vorgenommen werden.

1.1 Formale Systeme

Die Vorlesung befasst sich mit der „Mutter aller formalen Systeme“, der Prädikatenlogik, die wir alle bereits intuitiv verwenden: nämlich mit der Aussagenlogik und der Prädikatenlogik erster Ordnung.
Um zu beweisbaren Aussagen zu kommen, werden die Objekt- und die Meta-Ebene stets sorgfältig getrennt gehalten und nebenher gezeigt, was passieren kann, wenn man dies nicht tut. Außerdem wird eine genaue Semantik zur Interpretation der Formeln der Prädikatenlogik eingeführt.
Kalküle, das sind rein syntaktische Umformungen solcher Formeln, werden daraufhin untersucht, ob sie die Semantik genau bewahren, denn was würden sie nützen, wenn sie diese Eigenschaft (Korrektheit und Vollständigkeit) nicht hätten?
Da die Behandlung der Logik stets eine Reflexion auf das eigene Denken beinhaltet, werden immer wieder tiefer liegende Grundlagenprobleme gestreift, ohne sie allerdings allzu tief zu behandeln.
Anwendungen der Logik im automatischen Beweisen werden gestreift sowie die Programmiersprache PROLOG, die es erlaubt mit Logik zu programmieren („ausführbare Spezifikation“). Schließlich wird noch eine sehr effiziente Darstellung von booleschen Funktionen, nämlich die (binären) Entscheidungsdiagramme, die (O)BDDs, umrissen. Sie kommen bei der Behandlung sehr großer Zustandsräume, etwa im Chip-Entwurf zum Einsatz.

1.2 Algorithmentechnik

Es werden Algorithmen und Algorithmentechniken vertieft behandelt, einschließlich Komplexitäts- und Anwendungsuntersuchungen. Mögliche Problembereiche sind z.B. Langzahlarithmetik und Fouriertransformationen, Gruppen- und Matrixoperationen, Polynomarithmetik (eine sowie mehrere Variable), *Pattern Matching*, Kombinatorik, Probabilistische Algorithmen und Graphisch-geometrische Algorithmen.

1.3 Softwaretechnik

Ziel dieser Vorlesung ist es, das Grundwissen über Methoden und Werkzeuge zur Entwicklung und Wartung umfangreicher Software-Systeme zu vermitteln. Inhaltliche Themen: Projektplanung, Systemanalyse, Kostenschätzung, Entwurf, Implementierung, Validation und Verifikation, Software-Wartung, Software-Werkzeuge, Benutzerschnittstellen, Programmierumgebung und Konfigurationskontrolle.

1.4 Systemarchitektur

In der Vorlesung werden folgende Themen behandelt: Prozesse (*threads*) und Transaktionen, Adressräume und Domänen, Interaktionen in Form von Synchronisation (*events, semaphore, critical regions, monitors*), Kommunikation (*messages, rpc*) und Kooperation auf gemeinsamen Daten (*semaphores, shared memory*), temporäre und persistente Daten (*buffers, files*), Betriebsmittelverwaltungsarten (*physical, virtual resources*), lokale und verteilte Systeme (*local, distributed systems*), Zugriffsschutz und Zugangskontrolle (*access control, authentification*), mehrere Beispiele von strukturierten Systemen (*layered systems, client/server model, mikrokernel systems, realtime and/or mobile systems, agents*). In einigen der obigen Einzelthemen stecken Planungsprobleme (*scheduling*), die sowohl singulär als auch im Zusammenhang behandelt werden sollen.

1.5 Kommunikation und Datenhaltung

Verteilte Informationssysteme sind nichts anderes als zu jeder Zeit von jedem Ort durch jedermann zugängliche, weltweite Informationsbestände. Den räumlich verteilten Zugang regelt die Telekommunikation, die Bestandsführung über beliebige Zeiträume und das koordinierte Zusammenführen besorgt die Datenhaltung. Wer global ablaufende Prozesse verstehen will,

muss also sowohl die Datenübertragungstechnik als auch die Datenbanktechnik beherrschen, und dies sowohl einzeln als auch in ihrem Zusammenspiel.

1.6 Rechnerstrukturen

Die Kernvorlesung "Rechnerstrukturen" gibt einen Überblick über die Organisationsprinzipien und Hardware-Strukturen heutiger Rechner. Zunächst werden grundlegende Kenntnisse der Rechnertechnologie, des Rechnerentwurfs, der Leistungsmessung und Leistungsbewertung vermittelt. Danach werden Prozessortechniken von der grundlegenden von-Neumann-Architektur bis hin zu den heutigen superskalaren Mikroprozessoren vorgestellt. Bei der Organisation eines Mikroprozessors stehen die Nutzung der Parallelität auf der Befehlsebene durch Befehls-Pipelining und Superskalar-Prinzip sowie die Verringerung der Zugriffszeit auf den Hauptspeicher durch eine Hierarchie von Cache-Speichern im Vordergrund. Bei den Parallelrechnern werden auch andere Arten der Parallelität genutzt. Dazu gehören die Parallelität auf der Suboperationsebene bei den Vektorrechnern, die Datenparallelität bei den Feldrechnern und die Parallelität auf Prozessebene bei den Multiprozessoren. Alle drei Parallelrechnerklassen werden in ihren Grundprinzipien und anhand von Beispielen vorgestellt.
Neben der Erhöhung der Rechengeschwindigkeit spielt die Verbesserung der Zuverlässigkeit und Sicherheit eine große Rolle. Dies lässt sich durch Einführung von Redundanz auf den verschiedenen Architektur- und Implementierungsebenen erreichen. Es werden einfache Verfahren zur Fehlerdiagnose und Fehlermaskierung vorgestellt und im Hinblick auf die Verbesserung der Verfügbarkeit bewertet.

1.7 Echtzeitsysteme

Ziel der Vorlesung ist es, allgemeingültiges Grundwissen und Methoden des Fachgebiets "Prozessautomatisierung" zu vermitteln. Dabei stehen gleichrangig nebeneinander die Betrachtung der Aufgabe "Automatisierung" selbst, wie auch die geräte- und programmseitige Lösung und die Werkzeuge zur Unterstützung der Aufgabenbearbeitung. Es wird im Teilbereich "Geräte und Programmierung" auf dem Stoff anderer Grundvorlesungen aufgesetzt und hier der echtzeitspezifische Teil ergänzt.
Wesentliche Funktionsweisen heutiger Echtzeit- und Prozessrechner werden erläutert und auf deren Basis in die Probleme der Prozesssteuerung eingeführt. Weitere inhaltliche Themen sind: Automatisierung technischer Prozesse, Architektur von Automatisierungssystemen, Aufbau von Prozessrechensystemen, Prozessperipheriesysteme, Echtzeitprogrammierung, Projektierung von Automatisierungsanlagen.

1.8 Kognitive Systeme

Stichworte zum Inhalt: Repräsentationsverfahren und Methoden der Künstlichen Intelligenz, Signal-zu-Symbol-Übergang, Merkmalsauswertung und Mustererkennung, Repräsentation von Wissen, Suchverfahren, Ziehen von logischen Schlüssen, Maschinensehen, Sprachverstehen, Sensor-Roboter-Kopplung, Interpretation von sensorischen Eingaben in Handlungszusammenhängen, Aktionsplanungsverfahren, Akquisition von Wissen, maschinelles Lernen.
Ziel ist die Verdeutlichung der gemeinsamen methodischen Grundlagen für die Analyse und Implementierung technischer Systeme zur Lösung kognitiver Aufgaben.

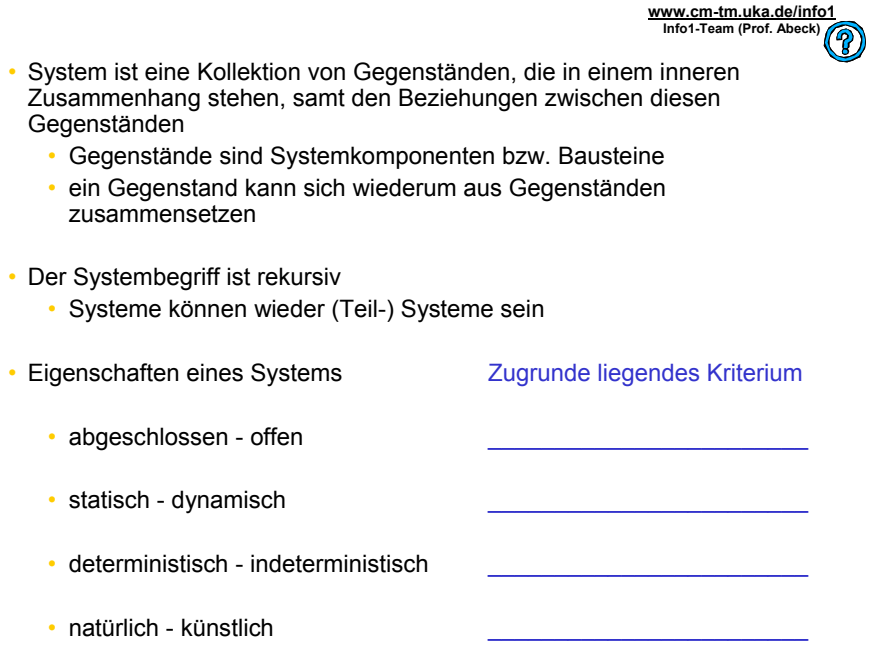
Die Beschreibungen sind u.a. Bestandteil des Studienleitfadens [FI04], der wertvolle Informationen zum Informatik-Studium an der Universität Karlsruhe (TH) enthält. Das Dokument sowie weitere relevante Informationen zu Forschung und Lehre können vom Fakultäts-Web unter www.ira.uka.de herunter geladen werden.

Eine noch detailliertere inhaltliche Gliederung der Informatik stellen die zahlreichen Vertiefungsfächer dar, die im letzten Studienabschnitt im Diplom-Studiengang angeboten werden. Bezüglich der Auflistung und Beschreibung dieser Fächer sei ebenfalls auf den Studienleitfaden [FI04] verwiesen.

2 INFORMATIKSYSTEME

Die Ausführungen zu den Teilgebieten der Informatik haben die zentrale Bedeutung von Informatiksystemen verdeutlicht. Daher soll der Begriff des Systems als erster und zentraler Grundbegriff detaillierter behandelt werden [Go97].

Weitere Grundbegriffe, die neben dem Begriff des Systems als zentral innerhalb der Informatik anzusehen sind, werden in der Kurseinheit GRUNDBEGRIFFE DER INFORMATIK eingeführt.



Interaktion 5: Systeme

Die Wirklichkeit und die diese repräsentierenden Modelle setzen sich aus zahlreichen Einzelkomponenten zusammen. Dabei stehen die Einzelkomponenten unter komplexen Wechselbeziehungen.
Die Systemmodellierung dient der Informatik zur Beschreibung und Umsetzung von Modellen. Auch in anderen Wissenschaften wird die Systemmodellierung angewendet. Die Informatik ist die Wissenschaft, die sich ganz grundsätzlich mit Systemen, deren Struktur, Beschreibung und Realisierung beschäftigt.

Im Zusammenhang mit den Systemen tritt mit der Rekursion zum ersten Mal ein Konzept in Erscheinung, das für die Informatik von besonderer Bedeutung ist. Rekursion bedeutet (allgemein formuliert), dass sich eine Sache wiederum teilweise aus der gleichen Sache zusammensetzt. In diesem Fall setzt sich ein System also wiederum aus einem System zusammen. Die Vorsilbe ("Teil-") drückt die wichtige Terminierung aus: Bei einem Teilsystem sollte es sich um ein „kleineres“ System als das Ursprungssystem handeln.

Systeme lassen sich gemäß der folgenden Eigenschaften klassifizieren:

- Die Eigenschaften ‘offen’ und ‘abgeschlossen’ beziehen sich auf das Kriterium der Systemgrenze. Ein abgeschlossenes System hat eine feste und zeitlich nicht veränderliche Systemgrenze, da dessen Bausteine keine Beziehungen zu den Gegenständen der Umgebung eingehen. Bei offenen Systemen besteht die Systemgrenze aus Schnittstellen des Systems (bzw. der Bausteine) zu den umgebenden Systemen.
  Das größte System aus der Sicht der Menschheit ist das Weltall. Es ist das einzige System, das keine Umgebung besitzt. Alle anderen Systeme besitzen eine Systemgrenze zu umgebenden Systemen.

- Ein statisches System liegt vor, falls das Kriterium der zeitlichen Veränderung bei dem Modell keine Rolle spielt. Bei dynamischen Systemen lassen sich die zeitlichen Veränderungen noch weiter klassifizieren (z.B. ändert sich die Menge der Bausteine oder nur Baustein-Eigenschaften).

- Die Eigenschaft des Deterministisch-Seins bezieht sich auf das Kriterium des Systemverhaltens.
  Liegt ein deterministisches System vor, so legt die Vergangenheit auch eindeutig das zukünftige Verhalten fest. Im nichtdeterministischen Fall bestehen also offensichtlich Einflussgrößen, die derzeit nicht bekannt und nicht zum System gehören.

- Aufgrund des Kriteriums der Systemart lassen sich natürliche Systeme (z. B. Biotop) und künstliche Systeme (z.B. Roboter) unterscheiden.

Die zu jeder Eigenschaft genannten Kriterien sind in Interaktion 5 zu ergänzen.

2.1 Systemtypen

Die in der Informatik entwickelten Systeme lassen sich in gewisse im Folgenden beschriebene Kategorien einordnen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

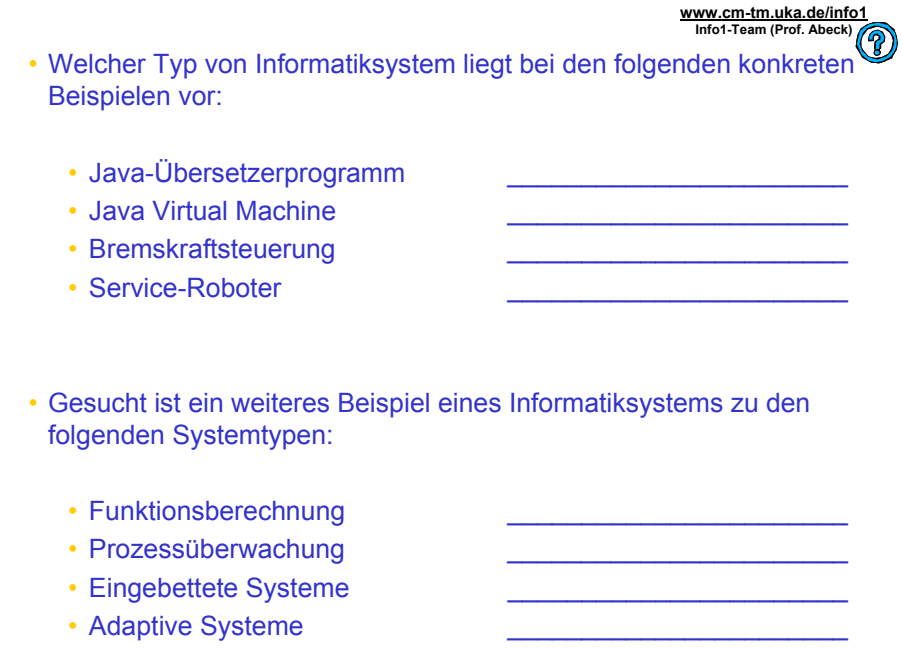
- Berechnung von Funktionen f: A -> B
- Prozeßüberwachung
  - Prozeß empfängt Daten von einem anderen Prozeß, verarbeitet diese (in Abhängigkeit der Prozeßhistorie) und sendet das Ergebnis an einen anderen Prozeß
- Eingebettete Systeme
  - Bausteine, die nicht der Datenverarbeitung dienen, sind am System-Verbund beteiligt
- Adaptive Systeme
  - eingebettetes System, das sich an Veränderungen der Wirklichkeit anpassen kann

Information 6: Typen von Informatiksystemen

Der in Information 6 vorgenommenen Unterteilung in Systemtypen liegt das Kriterium der durch die Systeme erbrachten Aufgaben bzw. der mit den Systemen verfolgten Ziele zugrunde. Die Aufzählung ist eine Hierarchie aufsteigenden Umfangs und wachsender Komplexität der zu lösenden Aufgabe.

- Funktionsberechnung: die elementarste der vier zu unterscheidenden Aufgabenstellungen, die als Teilaufgabe in allen drei folgenden Klassen von Aufgabenstellungen vorkommt.
- Prozessüberwachung: anstelle von Prozess kann man auch den Begriff des Systems verwenden. Die Prozesshistorie resultiert aus den lokal im Prozess gespeicherten Informationen. Durch die empfangenen Daten kann es zu einem Starten oder Halten des Prozesses kommen.
- Eingebettete Systeme: die Eigenschaft „eingebettet“ drückt aus, dass die Umgebung des Systems nicht DV-technisch ist (z.B. technische Apparaturen, betriebliche Organisationen, Menschen).

- Adaptive Systeme: die Eigenschaft „adaptiv“ ist eine spezielle Eigenschaft eingebetteter Systeme und besagt, dass sich das eingebettete System an Veränderungen der Wirklichkeit anpasst.



Interaktion 7: Beispiele von Informatiksystemen

In Interaktion 7 [BH03] sind Beispiele von Informatiksystemen genannt, die den in Information 6 genannten System-Kategorien zuzuordnen sind. Außerdem ist zu jeder der Kategorien mindestens ein weiteres System zu nennen.

2.2 Systemkonstruktion

Das zentrale Ziel, das sämtlichen Informatik-Fragestellungen zugrunde liegt, besteht in der Modellierung der Wirklichkeit (siehe Information 8). Zur Modellierung und deren Umsetzung des Modells sind geeignete Mittel erforderlich.

- Ziel
  - Modellierung der Wirklichkeit mit Informatik-Hilfsmitteln und Umsetzung des Modells

- Schritte der Konstruktion
  - Systemanalyse
  - Aufbau künstlicher Modellsysteme
  - Analyse des Modellsystems
    - ggf. Einsatz des Hilfsmittels der Simulation
  - Realisierung des Modells
    - falls das Modellsystem realisiert werden soll, ist eine Ist-Analyse und eine Soll-Analyse durchzuführen
    - Vorstufen: Simulation oder Prototyp

Information 8: Konstruktion von Informatik-Systemen

Zu beachten ist, dass die Wirklichkeit nicht nur die Realität umfasst, sondern auch gedachte Wirklichkeiten sein können, wie sie durch die Mathematik vorgegeben wird.

Diese natürlichen oder künstlichen Systeme sind im ersten Schritt, der Systemanalyse, im Hinblick auf ihre externen und internen Eigenschaften und Verhaltensweisen zu analysieren.
Die Ergebnisse der Systemanalyse fließen in ein so genanntes Modellsystem ein. Dieses Modellsystem ist dann im dritten Konstruktionsschritt im Hinblick auf dessen Wirklichkeitstreue zu überprüfen.

Hilfsmittel zur Überprüfung der Wirklichkeitstreue eines Modellsystems sind
- geschlossene mathematische Formeln,
- Methoden der theoretischen Informatik (z.B. formale Sprachen) oder
- Simulationen.

Die Simulation ist dabei die aufwendigste Überprüfungsform, da die Systemkomponenten mit deren inhaltlichen und zeitlichen Beziehungen auf dem Rechner realisiert werden müssen, damit die Eigenschaften (Systemziele) an Einzelfällen überprüft werden können. Die Simulation ist bereits eine Vorstufe zur Realisierung.

1. Entwurf des inneren Aufbaus des Systems
2. Ggf. rekursiver Entwurf der Teilsysteme
3. Festlegung der Verfahren zur Realisierung der Komponenten und deren (statischen und dynamischen) Beziehungen
4. Erstellung der Komponenten und Funktionen
5. Integration aller Komponenten und Funktionen
6. Integration des Systems in seine Umgebung

Information 9: Realisierung eines Informatiksystems

In Information 9 wird der letzte Konstruktionsschritt, die Realisierung, detailliert beschrieben. Diese besteht aus den angegebenen sechs Schritten.

Der Systementwurf enthält Rekursionen, sobald festgestellt wird, dass eine Komponente aufgrund ihrer Eigenschaften (Größe, Komplexität) als ein eigenes System zu betrachten ist.

Nach dem zweiten Schritt sollten die oben angegebenen Inhalte des Systementwurfs klar sein. Vor dem eigentlichen Erstellungsschritt müssen zunächst noch die Verfahren geklärt werden, was innerhalb des dritten Schritts erfolgt.

Der zentrale Schritt ist die Erstellung der Komponenten und Funktionen, also Schritt 4. Hierzu sind in den ersten drei Schritten diese Komponenten und Funktionen zu bestimmen und zu beschreiben. Schritte 1 und 2 können als Entwurfsschritte angesehen werden.

Die zwei verbleibenden Schritte 5 und 6 der Realisierung behandeln den Aspekt der Integration. Zum einen wird hier unter Integration der Zusammenbau der einzelnen Komponenten zu dem gewünschten Gesamtsystem verstanden, zum anderen die Einführung dieses Gesamtsystems in seine Umgebung. Mit der Integrationsaufgabe sind die Überprüfungen, ob das gewünschte Systemverhalten erzielt wird, auf das Engste verknüpft.

Mit der von der *Object Management Group* (OMG) standardisierten *Unified Modeling Language* (UML) steht heute in der Informatik eine Modellierungssprache zur Verfügung, durch die die Analyse und der Entwurf von Informatiksystemen geeignet unterstützt werden. Teile der Realisierungsarbeiten lassen sich dabei teilweise automatisieren, indem entsprechende Lösungen aus den Modellen heraus erzeugt werden [Oe01].

- Die Unified Modeling Language (UML) besteht aus Beschreibungselementen, mit denen im Grundsatz beliebige Systeme modelliert werden können
- Ein Haupteinsatzgebiet der UML besteht im Bereich der Modellierung von Informatiksystemen als Grundlage zur strukturierten Softwareentwicklung
- UML stellt graphische Beschreibungselemente bereit, die in festgelegter Form in gewissen Diagramm-Typen zum Einsatz kommen
- Beispiele von UML-Diagrammtypen
  - Anwendungsfalldiagramm zur Beschreibung der Umgebung (Systemgrenze) und des Einsatzzwecks eines Systems
  - Aktivitätsdiagramm zur Beschreibung von Tätigkeits-Abfolgen
  - Objekt-/Klassendiagramm zur Darstellung eines objektorientierten Programmentwurfs

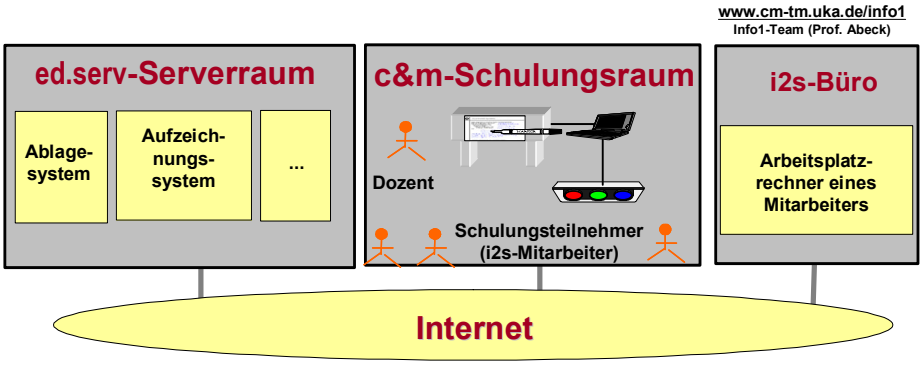
Information 10: Die Modellierungssprache UML

Die UML stellt graphische Beschreibungssymbole zur Verfügung, die zur Erstellung so genannter Diagramme dienen. Information 10 nennt drei der wichtigsten UML-Diagrammtypen. Während das Anwendungsfalldiagramm und das Objekt-/Klassendiagramm Beispiele statischer Diagrammtypen sind, beschreibt das Aktivitätsdiagramm dynamische Aspekte des zu modellierenden Systems.

2.3 Systembeispiel

Die Fragen, die sich im Zusammenhang mit der Konstruktion und der Nutzung von Informatiksystemen stellen, sollen an einem konkreten Beispielszenario und einem darin genutzten Systembeispiel aufgezeigt werden. Dabei kommen auch erste UML-Diagramme zur graphischen Darstellung gewisser Aspekte dieses Systembeispiels zum Einsatz.

Das behandelte Szenario ist aus dem Bereich der Aus- und Weiterbildung gegriffen und betrifft den Internet-basierten Wissenstransfer.



- Das Schulungsunternehmen c&m nutzt die vom IT-Dienstleister ed.serv angebotenen Dienste des Internet-basierten Wissenstransfers, um i2s einen Schulungsdienst anzubieten

- Das in diesem Szenario genutzte Informatiksystem ist ein durch das Internet verbundenes Verteiltes System
  - Server-Systeme: z.B. das von ed.serv betriebene Ablagesystem
  - Client-Systeme: z.B. der auf dem Arbeitsplatzrechner eines i2s-Mitarbeiters laufende Web-Browser

Information 11: Beispielszenario

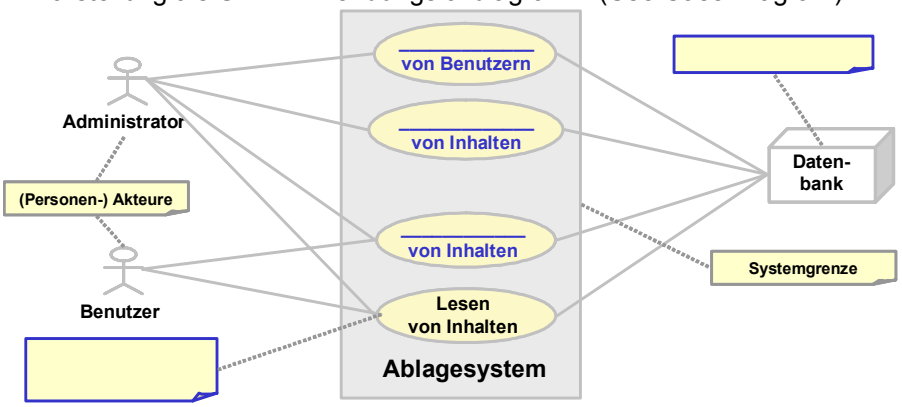
Die Situation, die durch die Abbildung in Information 11 [C&M-ASA] skizziert wird, lässt sich folgendermaßen zusammenfassen: Ein Schulungsunternehmen cooperation & more (c&m) nutzt zur Durchführung seiner Schulungen ein Internet-basiertes Wissenstransfersystem, durch das ein Dozent und die Schulungsteilnehmer flexibel auf die Wissensinhalte über das Internet zugreifen kann. Die IT-Infrastruktur und die darüber realisierten Datenverwaltungs- und Datenübertragungsdienste werden von einem Dienstleister educational.services (ed.serv) bereitgestellt.

Im Beispielszenario hält ein c&m-Dozent im Schulungsraum eine Schulung für einen Kunden, die Mitarbeiter des Unternehmens intelligent internet solutions (i2s) ab und erklärt einen komplexen Sachverhalt anhand von vorbereiteten Wissensinhalten, die er auf einer eingabesensitiven Oberfläche elektronisch annotiert. Neben den physisch anwesenden Teilnehmern im Schulungsraum verfolgen zahlreiche Mitarbeiter die Schulung von ihrem Arbeitsplatz aus über das Internet.

Das Beispielszenario zeigt auf, dass das genutzte Informatiksystem – das Wissenstransfersystem – ein verteiltes System ist und sich aus mehreren über das Internet vernetzten Einzelsystemen zusammensetzt.

- Das Ablagesystem ist ein Teil des Informatiksystems, das zur Ablage und gezielten Zugriff von Wissensinhalten dient
  - solche Systeme werden als Content-Management-Systeme bezeichnet

- Darstellung als UML-Anwendungsfalldiagramm (Use Case Diagram)



Interaktion 12: Modellierung des Ablagesystems

Eines dieser Einzelsysteme ist das Ablagesystem, das in Interaktion 12 näher betrachtet wird. Hierbei wird das von UML bereitgestellte Anwendungsfalldiagramm genutzt. Ein solches Diagramm legt die Systemgrenze des zu modellierenden Informatiksystems fest und beschreibt die Außenwelt des Systems durch Angabe der Rollen, die mit dem System in Beziehung stehen.

Der Anwender des zu modellierenden Systems wird üblicherweise als so genannter *Stickman* im Anwendungsfalldiagramm dargestellt. Im Beispiel treten zwei Anwender, der Administrator und der Benutzer auf. Systeme, die z.B. als ein Baustein gekennzeichnet werden können. Im Beispiel wurde nur die Datenbank als ein mit dem Ablagesystem in Beziehung stehendes System identifiziert.

Das Innenleben des so abgegrenzten Systems bilden die Anwendungsfälle bzw. *Use Cases*. Ein Anwendungsfall beschreibt eine Menge von Aktivitäten eines Systems aus Sicht seiner Akteure, die zu einem wahrnehmbaren Ergebnis für die Akteure führen und durch einen Akteur initiiert werden [Oe01].

Im konkreten Systembeispiel übernimmt der Administrator die Verwaltung der Benutzer, die auf das Ablagesystem zugreifen dürfen. Außerdem muss der Administrator in der Lage sein, alle im Zusammenhang mit den im System gehaltenen Inhalten notwendigen Arbeiten (Hochladen, Veröffentlichung, Suchen, Lesen) durchzuführen. Einem Benutzer werden die Suche und das Lesen von Inhalten ermöglicht, wobei ggf. nur ein Zugriff auf einen eingeschränkten Bereich vom Administrator gewährt wird.

- Hinter jedem Anwendungsfall verbirgt sich eine Folge von Aktivitäten, die geeignet durch ein Aktivitätendiagramm in UML beschrieben werden kann
  - Beispiel: Aktivitäten zum Anwendungsfall "Lesen von Inhalten"
    - beim Ablagesystem anmelden
    - Inhalt auswählen
    - Metadaten zum Inhalt lesen
    - Rechte-/Kostenaspekte akzeptieren
    - Inhalt herunterladen
    - ...

- Die Systemmodellierung mittels Anwendungsfalldiagrammen und die Anwendungsfälle beschreibende Aktivitätsdiagramme erfolgt in der ersten Phase der Systementwicklung
  - Analysephase
  - in der auf die Analysephase folgenden Entwurfsphase werden die von UML bereitgestellten Objekt-/Klassendiagramme eingesetzt

Information 13: Ausblick auf die nachfolgenden Modellierungsschritte

Das Anwendungsfalldiagramm steht ganz am Anfang der Systemmodellierung und ist daher Bestandteil der ersten Modellierungsphase, der Analysephase.

An die Aufstellung des Anwendungsfalldiagramms schließt sich eine detailliertere Beschreibung jedes Anwendungsfalls, wofür sich die von der UML bereitgestellten Aktivitätsdiagramme anbieten. Am Beispiel eines Anwendungsfalls, das Lesen von Inhalten, werden in Information 13 einige Aktivitäten genannt, die in diesem Anwendungsfall auftreten.

Dabei ist zu beachten, dass der Ablauf der Aktivitäten nicht zwingend streng sequentiell, sondern auch parallele oder bedingte Abläufe denkbar sind. Hierzu bietet das Aktivitätsdiagramm entsprechende Möglichkeiten an, auf die in späteren Kurseinheiten noch eingegangen wird.

Das Beispielszenario deutet bereits an, dass die von Informatikern entwickelten Systeme von ganz unterschiedlichen Anwendern (den Kunden des Informatikers) genutzt werden können. Nur wenn es dem Informatiker gelingt, die Kundenwünsche angemessen zu erfassen und in Systemmodellen umzusetzen, wird das von ihm entwickelte Informatiksystem eine Akzeptanz finden. Daher ist neben der fachlichen Qualifikation die Fähigkeit, die Sprache des Kunden zu verstehen, so wichtig.

3 TÄTIGKEITSPROFIL EINES INFORMATIKERS

Die übliche Bezeichnung des beruflichen Umfelds, in dem ein Informatiker tätig ist, wird als die "IT-Branche" bezeichnet. Die Abkürzung IT, die für Informationstechnik oder *Information Technology* steht, ist ein in der Praxis gebräuchlicher Begriff, der primär durch das Fachgebiet der Informatik geprägt ist.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Softwareentwicklung
  - Anwendungsentwickler, Koordination & Organisation
- Systementwicklung
  - Systemprogrammierer, Datenbank-Entwickler, Hardware-Spezialist
- Dienstleistungen
  - Berater, IT-Leiter, Administrator
- Sonstiges
  - z.B. Ausbilder, Vertriebs-Verantwortlicher

Information 14: Berufsfelder

In der IT-Branche existiert eine Vielzahl an Berufsbezeichnungen, durch die die jeweils zu erbringende Aufgabe bzw. die auszuführende Tätigkeit wiedergegeben wird. In [SG01] sind einige hundert solcher Berufsbezeichnungen (z.B. Anwendungsanalytiker, CAD-Berater, COBOL-Programmierer, DV-Ausbilder, Informationsmanager, Netzwerkbetreuer, Vertriebsberater) aufgeführt.

Information 14 nennt die wichtigsten Berufsfelder der IT-Branche. Einen Schwerpunkt der Tätigkeiten bildet die Entwicklung von Informatiksystemen, wobei hier die (anwendungs- bzw. systemorientierte) Softwareentwicklung eine besondere Bedeutung einnimmt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Technische Beratung / IT-Beratung / SAP-Beratung
- Marketing / Vertrieb
- Softwareentwicklung
- Netztechnik / Netzbetrieb
- Forschung & Entwicklung
- Anwendungsentwicklung
- Kundenservice

Information 15: Tätigkeitsschwerpunkte von Hochschulabsolventen

In Information 15 sind die Tätigkeitsfelder, in denen Hochschulabsolventen in der IT-Branche eingesetzt werden, gemäß ihrer Häufigkeit aufgelistet [SG01]. In den ersten zwei genannten Tätigkeitsbereichen, der Beratung und dem Vertrieb / Marketing, sind über die Hälfte der Hochschulabsolventen tätig. Ein weiteres großes Betätigungsumfeld betrifft die Software-Entwicklung und Anwendungsentwicklung. Es sei betont, dass von Beratern oder

Vertriebsbeauftragten ein fundiertes Wissen über die Entwicklung von Informatiksystemen und damit über Software-Entwicklung vorausgesetzt wird.

Ein nur relativ geringer Anteil an Hochschulabsolventen arbeitet im Bereich der Forschung und Entwicklung (F&E), wozu neben der Arbeit in F&E-Labors größerer IT-Unternehmen auch die Arbeit an den Hochschulen zählt. Der Anteil beläuft sich insgesamt auf ca. 15 Prozent, weniger als 10 Prozent der Informatiker arbeiten an Hochschulen.

- Analyse: Verständnis des zu lösenden Problems
  - Verständnis des Kunden und zukünftigen Anwenders
  - soziale Kompetenz, Gesprächsführung, in der Sprache des Kunden sprechen und schreiben können
- Entwurf: Systematische Entwicklung der Informatiklösung
  - fachliche Kompetenzen: Modellierung, Spezifikation, Wiederverwendung von Wissen (Entwurfsmuster)
- Implementierung: Umsetzung / Konstruktion der Lösung
  - ggf. Nutzung von Informatik-Werkzeugen (z.B. Code-Generatoren)
  - zu beachten: die eigentlichen Herausforderungen liegen in der Analyse und im Entwurf
- Wartung: Auslieferung, Betrieb, Betreuung, Anpassung (Customizing)
  - hier ist neben der fachlichen Kompetenz wiederum die soziale Kompetenz des Informatikers gefragt

Information 16: Aufgaben und erforderliche Kompetenzen

Die Aufgaben eines Informatikers lassen sich – falls diese unabhängig von einem bestimmten Tätigkeitsschwerpunkt formuliert werden sollen – am Entwicklungsprozess der Informatiklösung festmachen. Die in Information 16 [BH03] genannten Aufgabenschwerpunkte (Analyse, Entwurf, Implementierung, Wartung) entsprechen genau den Phasen des Softwareentwicklungsprozesses.

Ein guter Informatiker sollte neben fundierten fachlichen Kompetenzen insbesondere auch über ausgezeichnete soziale Kompetenzen verfügen. Diese betreffen die Kommunikationsfähigkeit (z.B. richtiger Umgang mit dem Kunden und den Kollegen im Team), die strategische Handlungskompetenz (z.B. Projektmanagement-Fähigkeiten) oder die Führungskompetenz (z.B. Leitung eines Projekts, Delegation von Aufgaben).

VERZEICHNISSE

Abkürzungen und Glossar

Abkürzung oder Begriff	Langbezeichnung und/oder Begriffserklärung
Anwendungsfall	UML-Element, das eine Menge von Aktivitäten eines Systems aus Sicht seiner Akteure beschreibt, die zu einem wahrnehmbaren Ergebnis für die Akteure führen und die durch einen Akteur initiiert werden [Oe01]. Englisch: *Use Case*
c&m	cooperation & more Im Szenario des Internet-basierten Wissenstransfers auftretendes Schulungsunternehmen.
ed.serv	eudcational.services Name des IT-Dienstleisters, der im Szenario des Internet-basierten Wissenstransfers auftritt.
F&E	Forschung und Entwicklung Eines der Tätigkeitsfelder eines Diplom-Informatikers.
i2s	intellligent internet solutions Name des als Kunden einer Schulung auftretenden Unternehmens, das im Szenario des Internet-basierten Wissenstransfers auftritt.
Internet-basierter Wissenstransfer	Beispielszenario, in dem Informationssysteme zur Unterstützung der Aufbereitung, der Vermittlung und der Aneignung von digitalisierten und über das Internet verfügbaren Wissensinhalten dienen.
IT	Informationstechnik Englisch: *Information Technology*
Rekursion	Liegt dann vor, wenn sich eine Sache wiederum teilweise aus der gleichen Sache zusammensetzt.
System	Kollektion von Gegenständen, die in einem inneren Zusammenhang stehen, samt den Beziehungen zwischen diesen Gegenständen.
UML	*Unified Modeling Language* Sprache, die aus graphischen Elementen besteht und zur semi-formalen Beschreibung von beliebigen Gegenständen (z.B. Software-Systeme oder Geschäftsbereiche) genutzt werden kann.
Wahlpflichtfach	Bezeichnung einer Art von Veranstaltung, die im Diplom-Informatik-Studiengang am Anfang des Hauptstudiums angeboten werden. Es werden die folgenden acht Wahlpflichtfächer angeboten: Formale Systeme, Algorithmentechnik, Softwaretechnik, Systemarchitektur, Kommunikation & Datenhaltung, Rechnerstrukturen, Echtzeitsysteme, Kognitive Systeme

Index

Informationen und Interaktionen

Literatur

[BH03] Bernd Brügge, Christian Herzog: Einführung in die INFORMATIK I, Vorlesungsfolien, TU München, 2003.

[C&M-ASA] Cooperation&Management, ALLGEMEINE STRUKTURIERUNGS- UND ARCHITEKTURPRINZIPIEN, Kursdokument zur Vorlesung "Kommunikation & Datenhaltung", http://www.cm-tm.uka.de/kud, Universität Karlsruhe (TH), C&M (Prof. Abeck)

[FI04] Fakultät für Informatik, Universität Karlsruhe (TH): Studium Informatik, www.ira.uka.de, 2004.

[Go97] Gerhard Goos: Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer Verlag 1997.

[Oe01] Bernd Oestereich: Objektorientierte Software-Entwicklung – Analyse und Design mit der Unified Modeling Language, Oldenbourg Verlag, 2001.

[Re94] Peter Rechenberg: Was ist Informatik? – Eine allgemeinverständliche Einführung, Carl Hanser Verlag, 1994.

[SG01] Joerg E. Staufenbiel, Birgit Giesen (Hrsg.): Berufsplanung für den IT-Nachwuchs, Staufenbiel Institut für Studien- und Berufsplanung, 2001.

PROGRAMMIERGRUNDLAGEN

Kurzbeschreibung

Diese Kurseinheit liefert die Grundlagen, einfache Programme in der Sprache Java zu entwickeln und auf einem Rechner ausführen zu lassen.

Schlüsselwörter

Programmerstellung, Grundsymbol, Variable, Anweisung, bedingte Anweisung, Schleife, Programmstruktur, Ein-/Ausgabe, Methode, Datentyp, Zahlen, Zeichen

Lernziele

1. Das grundsätzliche Vorgehen, das der Programmierung und der Ausführung eines Programms auf dem Rechner zugrunde liegt, wird verstanden.
2. Elementare Sprachelemente, wie Variablen, Zuweisungen, Anweisungen und Methoden sind bekannt und können zur Erstellung von Programmen genutzt werden.
3. Ein vollständiges und lauffähiges Java-Programm kann geschrieben und auf dem Rechner ausgeführt werden.

Hauptquellen

- Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

Inhaltsverzeichnis

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- PROGRAMM UND PROGRAMMIEREN
  - Vorgehen bei der Programmierung, Programmsyntax
- GRUNDLEGENDE BESTANDTEILE EINES PROGRAMMS
  - Variablen, Verzweigungen, Schleifen
- PROGRAMMSTRUKTUR
  - Grundstruktur, Ein-/Ausgabe, Methoden
- DATENTYPEN
  - Gleitkommazahlen, Zeichen

Information 1: PROGRAMMIERGRUNDLAGEN

1 PROGRAMM UND PROGRAMMIEREN

Programmieren bedeutet, ein Problem so zu beschreiben, dass es mittels eines Rechensystems gelöst werden kann. Das einem Programm zugrunde liegende Lösungsverfahren wird als Algorithmus bezeichnet. Die Sprache, in der der Algorithmus formuliert wird, ist eine Programmiersprache.

Auf den zentralen Informatikbegriff des Algorithmus wird in der Kurseinheit GRUNDBEGRIFFE DER INFORMATIK eingegangen. In der vorliegenden Kurseinheit werden die Programmiergrundlagen, die zur Erstellung einfacher Programme benötigt werden, am Beispiel der Sprache Java vermittelt [Mö03].

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Programmieren
  - ist eine kreative, anspruchsvolle und interessante Tätigkeit
  - sollte jeder Informatiker beherrschen, da hierdurch erst die Arbeitsweise eines Informatiksystems verstanden wird
- Programm
  - ist ein in einer Programmiersprache verfasster Algorithmus
  - ist auf einem Rechensystem ablauffähig
  - besteht aus Daten und Befehlen
- Sprachen — Beispiel
  - Höhere Programmiersprachen _____
  - Maschinennahe Sprachen _____
  - Modellierungssprachen _____
  - Datentransformationssprachen _____

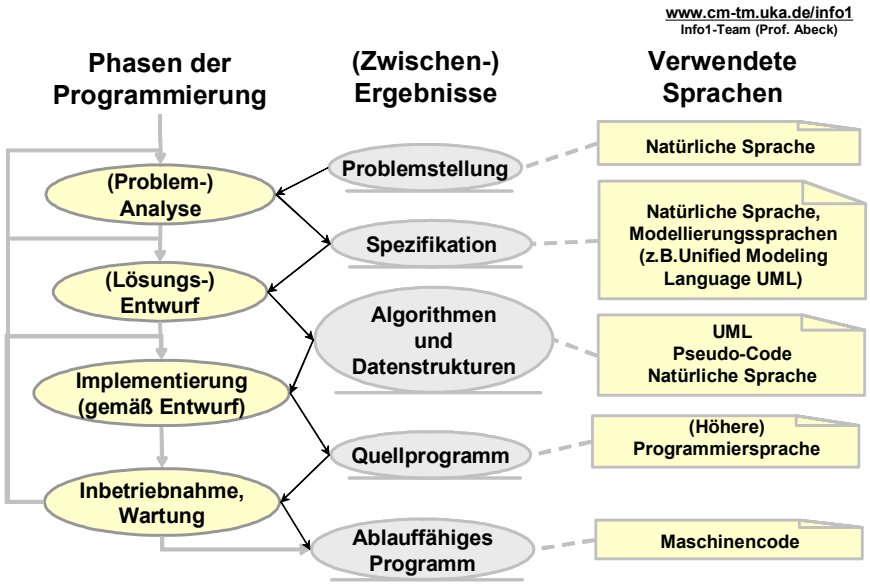
Interaktion 2: PROGRAMM UND PROGRAMMIEREN - Einführung der Begriffe

Das Programmieren umfasst sehr viel mehr als das reine Codieren einer Lösungsvorschrift in der vorgegebenen Programmiersprache. Das Finden und Analysieren einer Lösungsvorschrift sowie das Umsetzen in ein Programm sind kreative und anspruchsvolle Tätigkeiten. Die Fähigkeit, gute Programme zu schreiben, ist vergleichbar mit der Fähigkeit, gute Texte schreiben zu können.

Zum Programmieren ist eine Sprache notwendig, in der die Programme so erstellt werden können, dass diese von einem Rechner "verstanden" und bearbeitet werden können. Mit Programmiersprachen sind meist die so genannten höheren Programmiersprachen gemeint, wozu u.a. Java oder C gehören. Daneben existieren aber noch einige weitere Typen von Sprachen, die ebenfalls die Eigenschaft erfüllen, auf einem Rechner (zumindest teilweise) ausgeführt werden zu können. In Interaktion 2 ist zu jeder dieser Sprachtypen ein Beispiel zu nennen.

1.1 Vorgehen bei der Programmerstellung und -ausführung

Bei der Erstellung eines Programms kommen in der Praxis mehrere Sprachen zum Einsatz, wie in Information 3 [C&M-SM] aufgezeigt wird.



- Das Programmieren erfolgt iterativ, d.h. man springt häufig in eine der vorhergehenden Phasen zurück

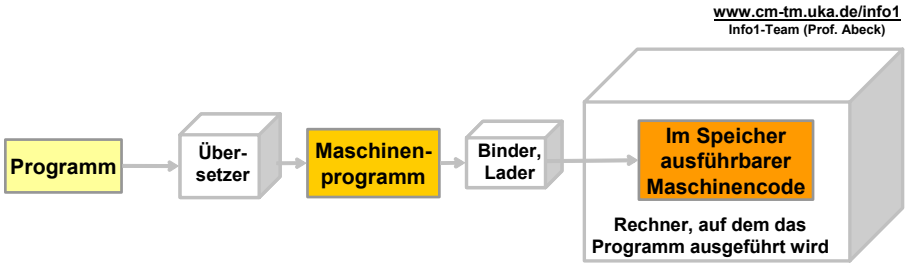
Information 3: Dem Programmieren zugrunde liegendes Vorgehen

Die Abbildung verdeutlicht, welche Phasen bei der Programmierung zu durchlaufen sind. Bei der Erstellung von größeren Programmen ("Programmieren im Großen") wird häufig der Begriff der Software-Entwicklung bzw. *Software Engineering* verwendet.

Durch den Ablauf wird aufgezeigt, dass vor der eigentlichen Implementierung – also der Beschreibung der Problemlösung in einer Programmiersprache – zunächst eine Analyse- und eine Entwurfsphase steht. Ist das Problem "klein", fallen diese Phasen entsprechend kurz aus. Da reale Informatikprobleme aber üblicherweise groß sind, kommt den beiden ersten Phasen in der Praxis eine erhebliche Bedeutung zu.

Neben den Phasen, die in Iterationen durchlaufen werden, sind in Information 3 auch die (Zwischen-) Resultate und die Sprachen aufgeführt, in denen diese Resultate vorliegen können.

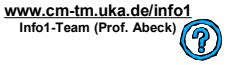
Neben der Programmiersprache Java wird heute intensiv die *Unified Modeling Language* (UML, [Oe01]) zur Analyse und zum Entwurf genutzt.



- Übersetzer, Binder und Lader sind auf einem Rechner ablauffähige Programme
  - der Übersetzer transformiert das Programm in ein bedeutungstreues Maschinenprogramm
  - der Binder wird benötigt, wenn das Programm mit anderen Programmteilen gebunden werden soll
  - der Lader lädt das Maschinenprogramm in den Speicher des (von-Neumann-) Rechners, auf dem das Programm ausgeführt wird

Information 4: Vom Quellprogramm zum ablauffähigen Programm

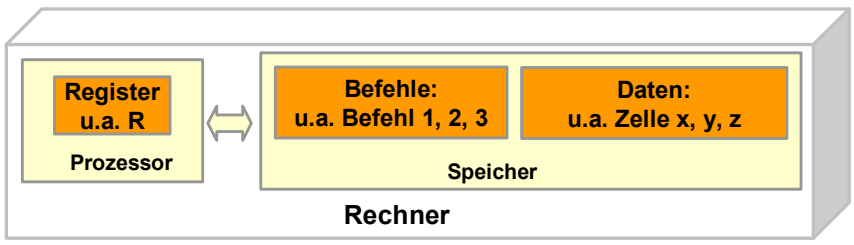
Information 4 zeigt, welche Schritte erforderlich sind, damit ein Programm auf einem (von-Neumann-) Rechner ablauffähig ist. Die hierzu benötigten Werkzeuge wie Übersetzer, Binder und Lader sind ebenfalls Programme.



- Folgender Programmbefehl soll ausgeführt werden:

```
z = x + y;
```

- Übersetzung in Maschinensprache (Pseudo-Code)
  Befehl 1: Lade Zelle x in ein Register R
  Befehl 2: Addiere Zelle y zu R
  Befehl 3: Speichere R in Zelle z



Interaktion 5: Ausführung eines Programmbefehls

In Interaktion 5 wird anhand eines einfachen Programmbefehls skizziert, wie die Umsetzung in ein Maschinenprogramm und die Ausführung auf dem Rechner erfolgt. In der Kurseinheit

GRUNDBEGRIFFE DER INFORMATIK wird näher ausgeführt, dass in der von-Neumann-Architektur ein Engpass, der sog. von-Neumannsche Flaschenhals, besteht.

Befehle und Daten sind jeweils binär codiert im Speicher abgelegt. Die Bits werden zu Bytes (8 Bit) gruppiert, Bytes wiederum zu Worten (2 oder 4 Byte) bzw. zu Doppelworten (2 Worte).

1.2 Programmsyntax

Der oben verwendete Programmbefehl

```
z = x + y;
```

gehört zum Sprachschatz der Programmiersprache Java. Die Festlegung des Sprachschatzes und damit des Aussehens – der Syntax – eines (Java-) Programms erfolgt durch eine so genannte Grammatik.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Die Syntax einer Sprache legt mittels Regeln fest, welche Sätze zu der Sprache gehören
- Eine Grammatik ist eine Menge von Syntaxregeln, durch die ein Sprachschatz beschrieben wird
  - Erweiterte Backus-Naur-Form (EBNF) ist eine weit verbreitete Schreibweise für Grammatiken
- Beispiel: EBNF zur Beschreibung einer Dezimalziffer

```
Ziffer = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
Zahl = Ziffer {Ziffer}.
```

- Unterscheidung von so genannten Terminal-Symbolen und Nichtterminal-Symbolen
  - "0" ... "9" Terminal-Symbole
  - Zahl, Ziffer Nichtterminal-Symbole

Information 6: Syntaxfestlegung mittels einer Grammatik

Zur Beschreibung einer Grammatik hat sich die nach den Informatikern John Backus und Peter Naur benannte Erweiterte Backus-Naur-Form, kurz EBNF, in der Praxis durchgesetzt. Wie Information 6 zeigt, lässt sich durch die EBNF beispielsweise die Syntax einer vorzeichenlosen Dezimalzahl kompakt beschreiben.

Die Backus-Naur-Form (BNF) besteht aus so genannten Terminal-, Nichtterminalsymbolen und der Metazeichenmenge {=, ., |}. In der Erweiterten BNF (EBNF) wurde die Metazeichenmenge um die Metazeichenmenge {(,), [,], {, }} ergänzt. Alle Metazeichen der EBNF werden im Folgenden beschrieben.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Metazeichen dienen zur Beschreibung der Grammatikregeln, durch die die zu einer Sprache gehörenden Sätze festgelegt werden

= trennt linke und rechte Regelseite

. schließt Regel ab

| trennt Alternativen
- Beispiel: x | y beschreibt: x, y

() klammert Alternativen
- Beispiel: (x | y) z beschreibt: xz, yz

[] wahlweises Vorkommen
- Beispiel: [x] y beschreibt: xy, y

{} 0-maliges bis n-maliges Vorkommen
- Beispiel: {x} y beschreibt: y, xy, xxy, xxxxy, ...

Information 7: Metazeichen der Erweiterten Backus-Naur-Form (EBNF)

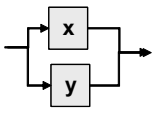
Die Metazeichen (siehe Information 7) geben die Syntax der EBNF-Regeln vor. Die Erklärungen und die zu einigen Zeichen gegebenen Beispiele beschreiben, was die Metazeichen bewirken (Semantik der Metazeichen).

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ein Syntaxdiagramm ist eine graphische Darstellungsform einer Grammatik

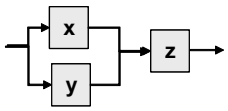
- Jedes EBNF-Metazeichen wird hierzu in Form einer graphischen Darstellung umgesetzt

- x | y



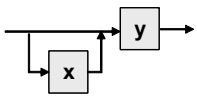
beschreibt: _____

- (x | y) z



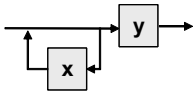
beschreibt: _____

- [x] y



beschreibt: _____

- {x} y



beschreibt: _____

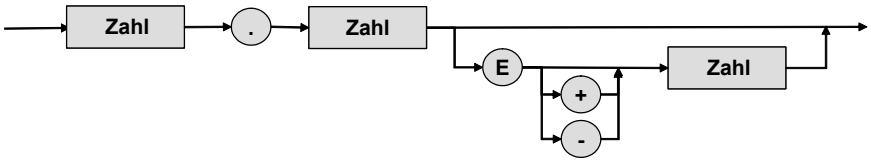
Interaktion 8: Syntaxdiagramm

Neben der textuellen Beschreibung besteht mit den Syntaxdiagrammen die Möglichkeit, EBNF-Regeln graphisch darzustellen. Wie in Interaktion 8 gezeigt wird, kann jedes Metazeichen durch eine graphische Darstellung umgesetzt und in einem Gesamtdiagramm zusammengestellt werden.

Die mittels eines Syntaxdiagramms beschriebene entspricht den alternativen Wegen, die durch das Diagramm (den Graphen) vorgegeben werden. In Interaktion 8 wird gefordert, dass zu jeder graphischen Darstellung die hierdurch beschriebenen (Teil-) Wörter anzugeben sind.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Unterscheidung der beiden Symbolarten in Syntaxdiagrammen
  - Terminale in Kreis-Symbolen
  - Nichtterminale in Rechteck-Symbolen



- Die untere EBNF-Grammatik ist so zu ergänzen, dass sie die Sprache beschreibt, die durch obiges Syntaxdiagramm dargestellt wird

```
Ziffer = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".

Zahl = Ziffer {Ziffer}.

Gleitkommazahl = _____
```

Interaktion 9: Syntaxdiagramm und Grammatik zu Gleitkommazahlen

Am Beispiel der Gleitkommazahlen wird in Interaktion 9 ersichtlich, wie sich die graphischen Elemente zu einem Syntaxdiagramm zusammenstellen lassen. In diesem Beispiel sind '.', 'E', '+' und '-' weitere Terminalsymbole, die zu den in Information 6 aufgeführten Symbolen hinzukommen. In der anzugebenden Grammatik ist neben Ziffer und Zahl ein drittes Nichtterminal-Symbol – hier Gleitkommazahl genannt – einzuführen.

In den folgenden Kapiteln wird die EBNF zur Beschreibung eines Ausschnitts der Sprache Java genutzt.

2 GRUNDLEGENDE SPRACHELEMENTE

Bereits das Schreiben einfacher Programme setzt voraus, dass gewisse Sprachelemente der verwendeten Programmiersprache bekannt sind. In Information 10 sind solche grundlegenden Sprachelemente am Beispiel von Java aufgelistet [Mö03].

- Grundsymbole
  - Schlüsselwörter, Zahlen, Zeichen
- Variablen und Zuweisung
  - Deklaration, Standardtypen
  - Initialisierung
- Ausdrücke
  - Arithmetische Ausdrücke
  - Boolesche Ausdrücke
- Anweisungen zur Ablaufsteuerung
  - Bedingte Anweisung
  - Schleife

Information 10: GRUNDLEGENDE SPRACHELEMENTE – Überblick

Die Grundsymbole liefern die Grundlage der Programmiersprache. Mittels dieser Symbole lassen sich die Datenelemente (z.B. Konstanten, Variablen) und Befehle (z.B. Zuweisungen, Schleifen) bilden. Grundsymbole umfassen die elementaren Sprachelemente. Hierzu gehören die zuvor beschriebenen Zahlenmengen.

Variablen sind Datenelemente, in denen (abhängig vom Typ) gewisse Werte gehalten werden können, die durch typspezifische Operationen geändert werden können. Variablen treten neben Zahlen und Zeichen in Ausdrücken auf. Je nach Typ des Resultats des Ausdrucks werden u.a. arithmetische und boolesche Ausdrücke unterschieden.

Durch Anweisungen lassen sich dynamische Aspekte in der Programmiersprache ausdrücken.

2.1 Grundsymbole

Ein Programm ist eine Folge von Symbolen, die nach bestimmten Regeln aufgebaut sind.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Namen bezeichnen Dinge (Variablen, Methoden, ...) in einem Programm

- Syntaktische Festlegungen
  1. erlaubte Zeichen: Buchstaben, Ziffern, "_" oder "$"
  2. erstes Zeichen entweder ein Buchstabe oder "_" oder "$"
  3. Groß-/Kleinschreibung ist signifikant
  4. beliebige endliche Länge

- Beispiele für syntaktisch korrekt gebildete Namen
  - x, _x, X1, first, FIRST, firstCourse

- Zu beachten: Nicht alle syntaktisch korrekten Namen machen Sinn
  - eine sinnvolle Namensgebung zählt zu den Eigenschaften, die ein gutes Programm ausmachen

Information 11: Namen

Ein zentrales Grundsymbol in Programmen sind die Namen (siehe Information 11).

Durch Namen legt der Programmierer fest, wie die im Programm eingeführten "Dinge" – z.B. Konstanten, Variablen, Klassen, Methoden – genannt werden sollen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Eine durchdachte und konsistente Namensgebung wird nur durch die konsequente Einhaltung von geeigneten Richtlinien erreicht

- Folgende Richtlinien gelten für alle INFORMATIK-I-Java-Programme (bitte bei den im Rahmen der Übungen zu erstellenden Programmen beachten):
  - Namenslänge
    - kurz im Falle von lokale, temporär verwendeten Namen (z.B. Laufvariablen)
    - eher etwas länger bei wichtigen Semantik-tragenden Namen
  - Worttrennung
    - Großbuchstaben zur Trennung von Wörtern in einem Namen
  - Sprache
    - englisch sowohl für Namen als auch für Kommentare

Beispiele:

i, j

course,
catalog

_____

// ordered by
// course number

Interaktion 12: Richtlinien

Für alle im Rahmen dieser Veranstaltung entwickelten Java-Programme gelten die im Folgenden eingeführten Richtlinien.

Namenslänge

Mit der Namenslänge ist ganz wesentlich die Verständlichkeit eines Namens verbunden. Grundsätzlich gilt, dass eine Sache durch einen längeren Namen verständlicher ausgedrückt werden kann. Zu lange Namen führen aber zu langen und schlecht lesbaren Programmen, weshalb kurze Namen zu bevorzugen sind. Grundsätzlich ist zu unterscheiden, um welche Art von Namen es sich handelt:

- Namen, die in einem kleineren, abgegrenzten Bereich des Programms genutzt werden, sollten kurz sein.
- Namen, die ein wichtiges Element der Problemlösung darstellen, sollten sprechend, aber nicht zu lang sein.

Worttrennung

Bedeutungstragende Namen setzen sich häufig aus mehr als einem Wort zusammen. Zur Trennung der Wörter bieten sich zwei Varianten an:

1. Trennung durch Großbuchstaben
2. Trennung durch Unterstreichungszeichen (_)

Für die im Rahmen dieser Veranstaltung zu erstellenden Programme kommt ausschließlich die erste Variante zur Anwendung, d.h. Trennung durch Groß-/Kleinschreibung (siehe Interaktion 12).

Sprache

Aus folgenden Gründen werden in diesem Kursbuch englische Namen gewählt:

- Englische Namen passen besser zu den ebenfalls englischen Schlüsselwörtern.
- Der mögliche Verbreitungsgrad der Programme ist größer.

Es gehört zum guten Programmierstil, zu allen nicht unmittelbar verständlichen Variablen durch einen Kommentar zusätzliche Erläuterungen in Form eines Kommentars im Programm vorzusehen. Die Syntax von Kommentaren wird später anhand konkreter Beispiele eingeführt. Kommentare werden in allen in diesem Kursbuch enthaltenen Programmen aus den oben genannten Gründen in englischer Sprache formuliert.

- Schlüsselwörter
  - bezeichnen die Sprachelemente und dürfen daher nicht als Namen verwendet werden
  - durchgängig klein geschrieben
  - Beispiele: if, for, boolean, int, static
- Zahlen
  - ganze Zahlen: dezimal (z.B. 10), hexadezimal (z.B. 0xC3A)
  - Gleitkommazahlen (z.B. 4.7E11)
- Zeichen
  - Buchstaben, Ziffern und Sonderzeichen, die in Unicode codiert sind
  - in einfache Hochkommata gesetzt
- Zeichenketten
  - in Hochkommata gesetzte Zeichen, z.B. "Informatik I"
  - " wird als \" geschrieben, z.B. "\"Informatik I\""

Information 13: Erste Sprachelemente

Durch die in Information 13 beschriebenen Schlüsselwörter einer Sprache wird die grundlegende Syntax einer Sprache angegeben. Die Sprache Java besteht aus den folgenden 48 Schlüsselwörtern:

1. abstract	2. boolean	3. break	4. byte	5. case
6. catch	7. char	8. class	9. const	10. continue
11. default	12. do	13. double	14. else	15. extends
16. final	17. finally	18. float	19. for	20. goto
21. if	22. implements	23. import	24. instanceof	25. int
26. interface	27. long	28. native	29. new	30. null
31. package	32. private	33. protected	34. public	35. return
36. short	37. static	38. super	39. switch	40. synchronized
41. this	42. throw	43. throws	44. transient	45. try
46. void	47. volatile	48. while		

Es fällt auf, dass alle Schlüsselwörter konsequent klein geschrieben sind. Da Java zwischen Klein- und Großschreibung unterscheidet, werden z.B. Int oder instanceOf vom Übersetzer nicht als Schlüsselwörter akzeptiert und führen zu einem Syntaxfehler.

Eine weitere Art von Grundsymbolen sind die Zahlen, wobei in Java ganze Zahlen und Gleitkommazahlen unterschieden werden. Ganze Zahlen können als (gewöhnliche) Dezimalzahlen oder als (der Bitdarstellung nahe liegenden) Hexadezimalzahlen geschrieben werden.

Zeichen und Zeichenketten sind in unterschiedliche Arten von Hochkommata (einfach bzw. doppelt) gesetzt. Die Zeichen sind intern durch Zahlen codiert, wobei der den ASCII-Code (*American Standard Code of Information Interchange*) erweiternde Unicode verwendet wird.

In Zeichenketten lassen sich gewisse Zeichen als Zeichenkombinationen (so genannte *Escape*-Sequenz) bestehend aus einem Schrägstrich (*Backslash* \, so genanntes *Escape*-Zeichen) und einem sich anschließenden Zeichen angeben.

2.2 Variablen und Zuweisung

Eine Variable stellt in Java ein grundlegendes Datenelement dar, das vor dessen Nutzung zunächst zu deklarieren, d.h. bekannt zu machen ist.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Variablen
  - sind Behälter (Container) von Werten
  - haben einen Namen
  - haben einen bestimmten Typ bzw. Datentyp, der die mögliche Wertemenge festlegt
  - müssen vor ihrer Verwendung deklariert (bekannt gemacht) werden

- Beispiel:
```
int max = 1000; // maximal number of something
```
  - Deklaration einer Variablen mit Namen max
  - max hat den Typ int (integer, d.h. ganze Zahl)
  - Compiler reserviert einen entsprechend großen Speicherplatz
  - max kann bei der Deklaration ein Wert (im Beispiel 1000) zugewiesen werden, der im Speicherplatz abgelegt wird
  - Kommentar (//) gibt zusätzliche Erläuterungen zur Variablen

Information 14: Variablen und deren Deklaration

Information 14 zeigt ein Beispiel eines Java-Programmausschnitts, durch den eine int-Variable mit dem Namen max deklariert wird und mit dem Wert 1000 belegt wird. Der sich anschließende Text maximal number of something ist ein Kommentar.

Durch die zwei Schrägstriche // wird festgelegt, dass ab dieser Stelle im Programm ein Kommentar folgt, der bis zum Zeilenende reicht. Eine alternative Form, Kommentare in einem Java-Programm anzugeben, ist die Verwendung von /* zur Angabe des Kommentaranfangs und */ zur Angabe des Kommentarendes.

Neben der Variablendeklaration bietet Java die Möglichkeit, Konstanten zu definieren. Die Konstanten kann man sich wie Variablen vorstellen, die bei ihrer Deklaration initialisiert werden und anschließend ihren Wert nicht mehr ändern.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Es bestehen folgende Standardtypen für ganze Zahlen in Java:
  - byte 8-Bit-Zahl $-2^7 \dots 2^7-1$ (-128 ... 127)
  - short 16-Bit-Zahl $-2^{15} \dots 2^{15}-1$ (-32768 ... 32767)
  - int 32-Bit-Zahl $-2^{31} \dots 2^{31}-1$ (-2.147.483.648 ... 2.147.483.647)
  - long 64-Bit-Zahl $-2^{63} \dots 2^{63}-1$

- Die Längen der einzelnen Typen gelten plattformunabhängig

- Ganze Zahlen sind in Java grundsätzlich vorzeichenbehaftet

- Der Datentyp bestimmt
  - die Menge von Werten, die zu diesem Typ gehören
  - die Menge von Operationen, die mit den Werten des Typs ausgeführt werden dürfen

Information 15: Standardtypen für ganze Zahlen und Datentyp

Wie die in Information 15 enthaltene Auflistung der Standardtypen von den in Java angebotenen ganzen Zahlen aufzeigt, reserviert der Compiler für eine int-Zahl grundsätzlich – d.h. auf jedem Rechner mit beliebigem Betriebssystem – 32 Bit Speicher. Hierdurch wird bestimmt, welche Zahlenwerte in der int-Variablen aufgenommen werden können. Je nach benötigter Wertemenge können statt int die Standardtypen byte oder short (bei kleineren Werteräumen) und long (im Falle eines größeren Werteraums) als Typ vereinbart werden.
Im Gegensatz zu anderen Programmiersprachen (wie z.B. C) kennt Java keine vorzeichenlosen (*unsigned*) Zahlen.

Die zu den ganzen Zahlen kennen gelernten Typen sind Beispiele für Datentypen. Datentypen gehören zu den fundamentalen Konzepten einer modernen Programmiersprache. Dieses Konzept ermöglicht dem Compiler eine statische Typprüfung, in der festgestellt wird, ob eine Variable einen zulässigen Wert enthält und ob die auf einer Variablen ausgeführte Operation aufgrund des deklarierten Datentyps erlaubt ist. Zahlreiche Programmierfehler lassen sich auf diese Weise aufdecken.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Eine Zuweisung ist eine Anweisung, durch die einer (deklarierten) Variablen der Wert eines Ausdrucks zugewiesen wird
- Beispiele:

```
int x = max - 1;
```

```
int x;
x = max - 1;
```

  - int x Deklaration der Variablen x
  - '=' Zuweisungszeichen
  - max - 1 Wert, der x zugewiesen wird
  - ; Abschluss der Anweisung

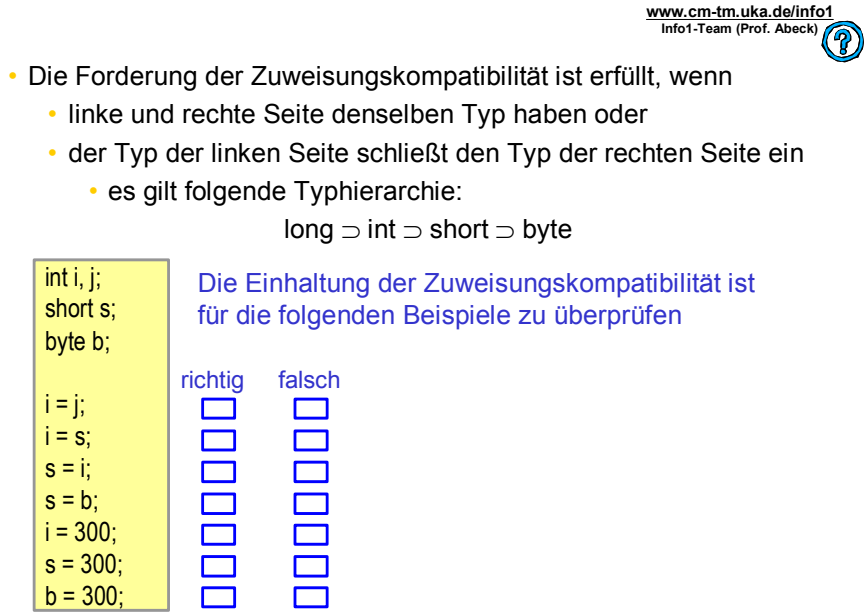
- Die in den Kommentaren beschriebenen Zuweisungen sind zu ergänzen

```
_____ // assign value of x to long variable l

_____ // assign 250 to short variable s

_____ // assign value of s minus value of x to l
```

Interaktion 16: Zuweisung

Ein grundlegender Befehl im Zusammenhang mit Variablen ist die Zuweisung (*Assignment*). Anstelle von Befehl wird auch der Begriff der Anweisung (*Statement*) verwendet. Wie die beiden Programmausschnitte in Interaktion 16 zeigen, kann einer Variablen bei ihrer Deklaration direkt ein Wert zugewiesen werden. Alternativ können die Deklaration und die Zuweisung in zwei getrennten Befehlen erfolgen.
Als Zuweisungsoperator wird in Java das Gleichheitszeichen '=' verwendet. Es ist zu beachten, dass die Zuweisungsbeziehung keine Gleichheitsbeziehung darstellt.

Zu den in Interaktion 16 angegebenen Kommentare sollen die entsprechenden Zuweisungen als Java-Befehle angegeben werden.



Interaktion 17: Zuweisungskompatibilität von Variablen ganzzahliger Typen

Nicht jeder Variablen kann ein beliebiger Wert zugewiesen werden, weshalb eine so genannte Zuweisungskompatibilität erforderlich ist. Die Kompatibilitätsforderung ist dann erfüllt, wenn der Typ der rechten Seite den Typ der linken Seite einschließt, d.h. die Wertemenge, die der Typ der rechten Seite beschreibt, umfasst die Wertemenge, die durch den Typ der linken Seite beschrieben ist.

Aufgrund der unterschiedlichen Wertemengen (siehe Information 15 weiter oben), die durch die verschiedenen ganzzahligen Typen festgelegt werden, ergibt sich die in Interaktion 17 angegebene Typhierarchie. Anhand der Beispiele ist zu überprüfen, ob die Zuweisungskompatibilität eingehalten oder verletzt wird.

Zahlkonstanten werden in Java (unabhängig von deren Wert) immer als int angenommen. Allerdings akzeptiert der Übersetzer eine Zuweisung einer Zahl zu einer short- bzw. byte-Variablen, falls der Wert von dieser Variablen aufgenommen werden kann.

2.3 Ausdrücke

Mit Variablen und Konstanten lässt sich mittels der zu den jeweiligen Typen definierten Operatoren rechnen. Im Folgenden werden für ganzzahlige und boolesche Typen die in Ausdrücken formulierbaren Berechnungsformeln vorgestellt.

2.3.1 Arithmetische Ausdrücke

Mit dem Ausdruck y + 1 wurde bereits ein erstes einfaches Beispiel eines arithmetischen Ausdrucks eingeführt. Information 18 zeigt einen weiteren arithmetischen Ausdruck.

- Berechnung eines numerischen Wertes aus Variablen und Konstanten mittels arithmetischer Operatoren
  - Beispiel: -x / (y - 1)

- Syntax in Form von EBNF-Regeln

```
Expression = Operand {BinaryOperator Operand}.
Operand = [UnaryOperator] (Variable | Number | "(" Expression ")" ).
```

- Standard-Operatoren
  - binär: + - * / %
  - unär: + -

- Spezielle Operatoren in Java
  - Shift: x << y x >> y
  - Inkrement: x++ ++x
  - Dekrement: x-- --x
  - Zuweisungsoperator: x += y x -= y x*= y x /= y x %= y

Information 18: Arithmetischer Ausdruck

Die Syntax eines arithmetischen Ausdrucks lässt sich durch die zwei angegebenen EBNF-Regeln einfach festlegen.

Neben den bekannten binären (2-stelligen) und unären (1-stelligen) Standardoperatoren, für die die bekannten Vorrangregeln gelten, bietet Java eine Reihe von Spezialoperatoren an. Durch die *Shift*-Operationen lassen sich im Falle von zeitkritischen Anwendungen schnelle Multiplikationen und Divisionen mit Zweierpotenzen durchführen.

Inkrement (x++ entspricht x = x+1) bzw. Dekrement (x-- entspricht x = x-1) und der Zuweisungsoperator (x += y entspricht x = x + y) sind Kurzschreibweisen, die von Java angeboten werden. Inkrement bzw. Dekrement sollten aufgrund der komplizierten Semantik, die bei deren Verwendung in einem arithmetischen Ausdruck entstehen, möglichst nur als eigenständige Anweisung auftreten.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
// Example: result type and type cast
short s;
int i;
long l;
... l + 1 ...                  // long
... s + 1 ...                  // int (1 is of type int)
s = (short)(s + 1);            // type cast required
```

- Zu beachten: Zahlenkonstanten (im Beispiel: 1) sind vom Typ int
  - Operation mit short-Variablen liefert eine int-Variable als Ergebnis

- Durch eine Typkonversion wird der Typ eines Ausdrucks in den gewünschten Typ umgewandelt
  - im Beispiel: es werden die ersten beiden Bytes des int-Werts abgeschnitten

Information 19: Ergebnistyp eines arithmetischen Ausdrucks und Typkonversion

Im Zusammenhang mit Ausdrücken stellt sich die Frage, von welchem Typ das Ergebnis eines ganzzahligen Ausdrucks ist. Hierzu besteht die folgende Festlegung: Wenn mindestens ein Operand vom Typ long ist, so ist der Ergebnistyp long, sonst ist der Ergebnistyp int.

Das bedeutet, dass eine Zuweisung des Ergebnisses eines arithmetischen Ausdrucks an eine Variable mit dem Typ short oder byte immer eine so genannte, im Programm in Information 19 enthaltene Typkonversion erforderlich macht.

2.3.2 Boolesche Ausdrücke

Der Datentyp boolean ist ein weiterer elementarer Datentyp, der insbesondere für die im folgenden Abschnitt behandelte Programmablaufsteuerung benötigt wird.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Typ boolean ist neben den bereits kennen gelernten ganzzahligen Typen (long, int, short, byte) ein weiterer wichtiger elementarer Datentyp
  - kann die Werte true oder false annehmen
  - Operatoren: && || !

- Ein boolescher Wert kann aus einem Vergleich von Zahlenwerten resultieren
  - Operatoren: == != > < >= <=

```
// Example: data type boolean
int x = 1;
boolean p, q;
p = false;
q = 0 < x;                     // true or false? _______
p = (p || q) && x < 10;        // true or false? _______
```

Interaktion 20: Datentyp boolean

Der Einsatz der booleschen Operatoren (und &&, oder ||, nicht !) sowie der Vergleichsoperatoren, die einen booleschen Wert als Ergebnis liefern, lässt sich anhand des Programmausschnitts in Interaktion 20 nachvollziehen.

Die den Ausdrücken zugrunde liegenden algebraischen Grundlagen werden in einer späteren Kurseinheit behandelt.

2.4 Anweisungen zur Ablaufsteuerung

Die Steuerung des Kontrollflusses gehört zu den wesentlichen Bestandteilen eines Algorithmus. Mit der bedingten Anweisung und der Schleife werden zwei grundlegende Anweisungen eingeführt und am Beispiel von Java anhand der if-Anweisung und der while-Anweisung präzisiert.

2.4.1 Bedingte Anweisung

In Abhängigkeit einer als boolescher Ausdruck formulierten Bedingung wird eine von zwei alternativen Anweisungen ausgeführt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ziel: Eine Anweisung soll nur unter bestimmten Bedingungen ausgeführt werden
- Syntax (Erweiterte Backus-Naur-Form EBNF)

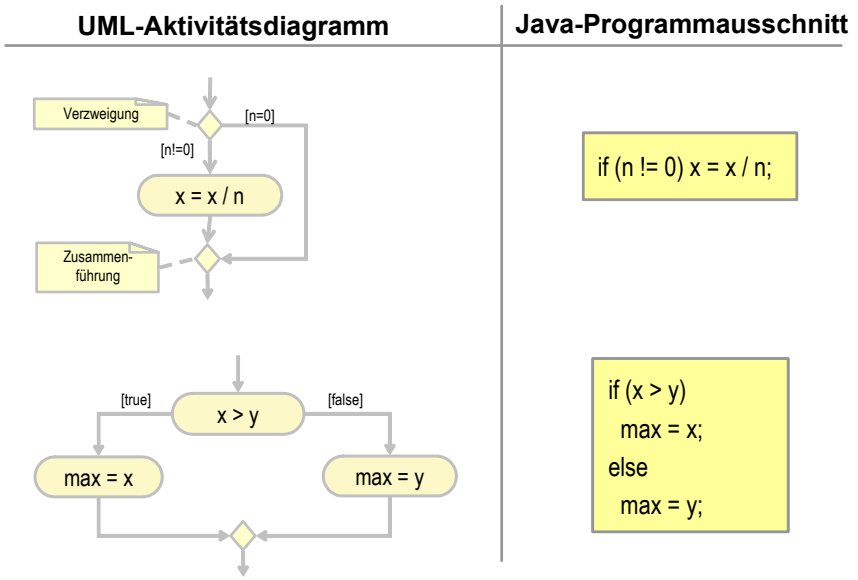
```
IfStatement = "if" "(" Expression ")" Statement ["else" Statement].
```

- Nichtterminale: _____
- Terminale: _____
- Wie heißt die dritte Art von EBNF-Zeichen? _____
- Das Nichtterminal Expr ist ein Ausdruck (Expression), dessen Auswertungsergebnis als boolescher Wert wahr (true) oder falsch (false) interpretiert wird

Interaktion 21: Bedingte Anweisung

In Interaktion 20 ist die Syntax einer bedingten Anweisung in Form einer EBNF-Regel angegeben.

Bedingte Anweisungen (IfStatement) werden (wie bedingte Ausdrücke) in ihrer Bedeutung auf die Fallunterscheidung zurückgeführt. Hat der nach der if-Anweisung folgende Ausdruck (Expression) den Wert true, so wird die Anweisung (Statement) im then-Zweig (der Anweisungsblock direkt nach der Bedingung) ausgeführt, andernfalls der else-Zweig.



Information 22: Beispiele

In der *Unified Modeling Language* (UML) lassen sich Bedingungen in einem Aktivitätsdiagramm darstellen. UML-Aktivitätsdiagramme, die Programmabläufe beschreiben, werden auch als Flussdiagramme bezeichnet.

Wie die beiden Beispiele in Information 22 zeigen, stehen zwei Möglichkeiten zur Verfügung: Im ersten Fall wird sowohl die Zusammenführung als auch die Verzweigung des Kontrollflusses explizit durch eine Raute (*Diamond*) angegeben, im zweiten Fall werden die Bedingungen direkt an die ausgehenden Transitionen der Aktivität geschrieben, die den booleschen Ausdruck berechnet. Die Beispiele verdeutlichen, dass die zweite Variante besser lesbar ist und eine einfachere Umsetzung in ein Java-Programm ermöglicht.

- Die Syntax von Java erlaubt nur eine Anweisung im then-Zweig (true-Zweig) und im else-Fall (false-Zweig)
  - gibt die Syntax der Programmiersprache vor
  - stellt aber keine grundsätzliche Einschränkung dar, weil
    - Anweisungsfolgen lassen sich durch Setzen in geschweifte Klammern {...} zu einem Block zusammenfassen
    - ein Block wird als eine Anweisung betrachtet

```
if (x < 0) {
  negNumbers++;
  Out.print(-x);
} else {
  posNumbers++;
  Out.print(x);
}
```

Was leistet der Programmausschnitt?

_____

_____

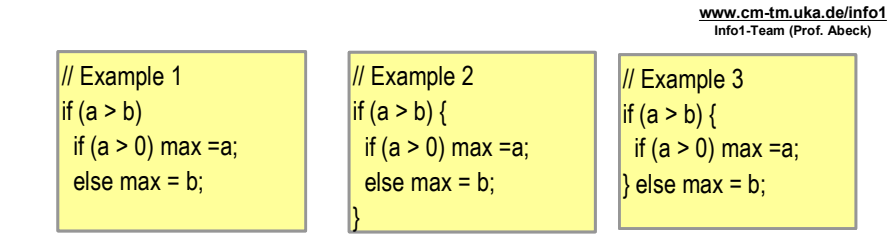
_____

Interaktion 23: Anweisungsblöcke

Sollen im then- oder else-Zweig mehrere Anweisungen ausgeführt werden, so sind diese aufgrund der Syntax-Vorschrift zu jeweils einem Anweisungsblock zusammen zu führen.

Interaktion 23 zeigt ein Beispiel eines Programmausschnitts, dessen Ergebnis informell zu beschreiben ist.

An den Beispielen wird deutlich, dass die Lesbarkeit eines Java-Programms durch entsprechende Einrückungen erhöht wird. Als Einrückung wurden zwei Leerzeichen vorgesehen. Wie das erste Beispiel aus Information 22 zeigte, können kurze if-Anweisungen auch in einer Zeile geschrieben werden.



- Problem: Zuordnung des else-Zweigs bei mehreren vorausgehenden verschachtelten if-Anweisungen
- Lösung: Der else-Zweig gehört immer zur unmittelbar vorausgehenden if-Anweisung
  - Klammern in Beispiel 2 damit überflüssig
- Zuordnung des else-Zweigs zu einer anderen if-Anweisung erfolgt über Zusammenführung von Anweisungsfolgen zu einem Block
  - siehe Beispiel 3

Information 24: Hängender else-Zweig

Mehrere if-Anweisungen können verschachtelt werden, d.h. es kann in einem then-Zweig oder else-Zweig eine weitere if-Anweisung erfolgen. Eine if-Anweisung im then-Zweig kann zu einem so genannten *dangling else* – also einem "in der Luft" hängenden else-Zweig führen. Die Problematik und deren Lösung in Java verdeutlicht Information 24.

2.4.2 Schleife

Die Schleife ist ein Sprachelement, durch das eine mehrmalige Ausführung von Programmteilen ermöglicht wird.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ziel: Gewisse Berechnungen sollen wiederholt ausgeführt werden, bis eine bestimmte Bedingung eintritt
- Dieses Ziel wird u.a. durch die while-Schleife (Abweisschleife) erreicht
- Syntax der while-Schleife
  - while (<Schleifenbedingung>) <Schleifenrumpf>
  - als EBNF-Regel
    - durch die Bildung eines Blocks können beliebig viele Anweisungen den Schleifenrumpf bilden

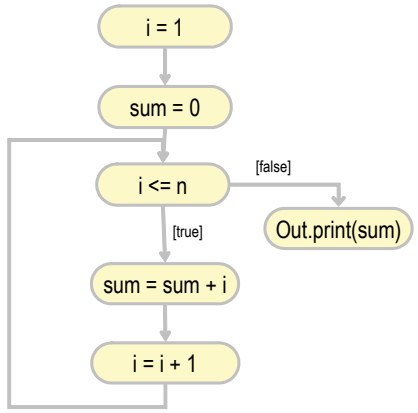
```
WhileStatement = "while" "(" Expression ")" Statement.
```

Information 25: Schleifen

Es lassen sich im Wesentlichen drei Arten von Schleifen unterscheiden: Abweisschleife, Durchlaufschleife, Zählschleife. In dieser Kurseinheit soll zunächst die Abweisschleife behandelt werden (siehe Information 25), die sich durch die while-Anweisung ausdrücken lässt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

UML-Aktivitätsdiagramm



Java-Programmausschnitt

```
i = 1; sum = 0;
while (i <= n) {
  sum = sum + i;
  i = i + 1;
}
Out.print(sum);
```

- Was wird durch den Programmausschnitt berechnet?

Interaktion 26: Beispiel zur while-Anweisung

Ein Beispiel eines Programmausschnitts mit einer while-Schleife zeigt Interaktion 26.

Beim Programmieren einer Schleife ist sicher zu stellen, dass die Schleifenbedingung auch tatsächlich nach endlich vielen Durchläufen durch den Schleifenrumpf nicht mehr erfüllt wird und die Schleife verlassen wird. Andernfalls würde der Rechner endlos die Anweisungen in der Schleife ausführen und das Programm würde nicht terminieren.

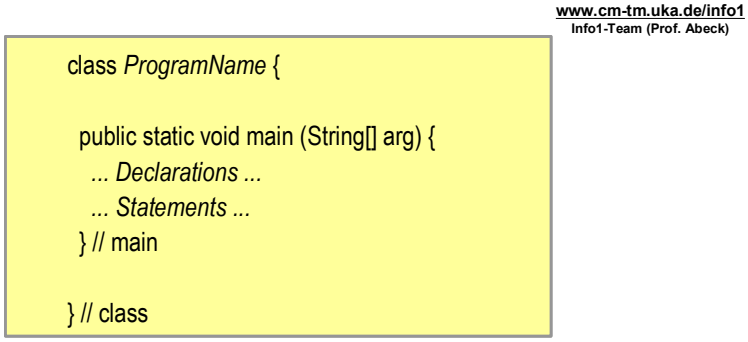
Auf das Problem der Schleifenterminierung sowie die oben erwähnten weiteren Schleifenarten werden in der Kurseinheit IMPERATIVE PROGRAMMIERUNG detailliert eingegangen.

3 PROGRAMMSTRUKTUR

Die bislang behandelten Java-Beispiele stellen nur jeweils Ausschnitte aus einem Java-Programm dar. In diesem Abschnitt sollen nun alle noch fehlenden Aspekte zusammengetragen werden, damit vollständige, auf dem Rechner ausführbare Java-Programme erstellt werden können [Mö03].

3.1 Grundstruktur

Jedes Java-Programm folgt einer ganz bestimmten Grundstruktur, die vom Compiler verlangt wird, um ein Programm übersetzen zu können.



- Die Grundstruktur ist vorläufig als ein einfacher syntaktischer Rahmen zu verstehen
  - kursiv geschriebene Teile sind vom Programmierer zu ersetzen
  - zunächst gibt es nur eine Klasse (class *ProgramName*), die das gesamte Programm beinhaltet
  - main() ist eine Methode der Klasse *ProgramName*

- Namens-Schreibweise
  - Klassennamen beginnen mit einem Großbuchstaben
  - Methodennamen beginnen mit einem Kleinbuchstaben

Information 27: PROGRAMMSTRUKTUR – Grundstruktur eines Java-Programms

Diese Grundstruktur zeigt Information 27. Das gesamte Programm besteht aus einer Klasse class, die den Programmnamen trägt. Das Klassen-Konzept von Java wird erst in einer späteren Kurseinheit eingeführt.

Zum Klassenkonzept muss zunächst nur bekannt sein, dass in Klassen so genannte Methoden auftreten. Methoden sind Rechenvorschriften, denen Parameter übergeben werden können und die einen Rückgabewert abliefern können. Die Methode main() ist eine in Java ausgezeichnete Methode, da diese den Einstiegspunkt in das Programm markiert.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
class FirstProgram {
  public static void main (String[] arg) {
  int v1, v2, max;                    // declare variables
  v1 = 17; v2= -9;                    // assign variables
  max = v1;                           // determine maximum
  if (max < v2) max = v2;
  }
}
```

- Zu klären
  - Ausgabe des Ergebnisses bzw. Eingabe von Werten
  - Ausführung des Programms auf dem Rechner

Information 28: Erstes vollständiges Java-Programm

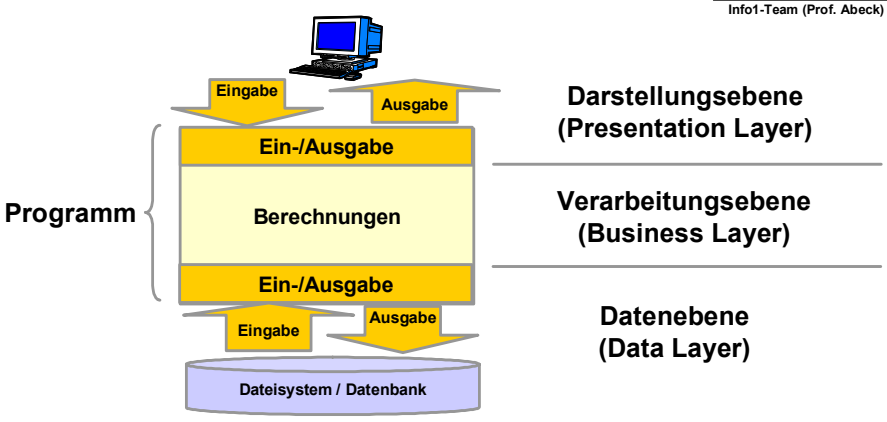
Information 28 führt ein erstes vollständiges Java-Programm ein, durch das das Maximum max von zwei gegebenen Variablenwerten v1 und v2 ermittelt wird.

Die in Information 28 genannten Punkte, die zwingend zu klären sind, betreffen die nachfolgend behandelte Ein-/Ausgabe.

3.2 Ein-/Ausgabe und Programmausführung

Ein wichtiger technischer Aspekt, der bislang noch nicht im Detail betrachtet wurde, betrifft die Frage, wie einem Programm Daten zugeführt (Eingabe) bzw. aus dem Programm Daten bereitgestellt (Ausgabe) werden können.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)



- Ein Programm erhält die zu verarbeitenden Daten von seiner Umgebung und gibt Daten an die Umgebung ab
  - die Umgebung besteht aus einer persistenten Datenhaltung und der Schnittstelle zum Benutzer
  - das Ergebnis ist eine Drei-Schichten-Architektur
    - Daten – Verarbeitung – Darstellung

Information 29: Ein-/Ausgabe

Wie die Abbildung in Information 29 zeigt, besteht die Umgebung eines Programms, mit der Daten ausgetauscht werden, zum einen aus der Benutzerschnittstelle mit der Tastatur als

Eingabegerät und dem Bildschirm als Ausgabegerät. Zum anderen kann das Programm mit dem Dateisystem (oder auch mit einem Datenbanksystem) Daten austauschen, indem es auf die Datei (bzw. Datenbank) lesend oder schreibend zugreift.

Mit den beiden Ein-/Ausgabeformen sind zwei Ebenen, die Daten- und die Darstellungsebene, verbunden. Die durch das Programm realisierte Funktionalität stellt die eigentliche Verarbeitungsebene dar. Insgesamt liefern die drei Ebenen eine Architektur, die auf beliebige Software-Systeme angewendet werden kann.

3.2.1 Klassen In und Out

Nachfolgend wird zunächst aufgezeigt, wie Ausgaben zur Benutzerschnittstelle (Darstellungsebene) realisiert werden können.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
if (x < 0) {
  negNumbers++;
  Out.print(-x);
} else {
  posNumbers++;
  Out.print(x);
}
```

- Out.print() ist ein Befehl zur Ausgabe von Daten (hier -x bzw. x)
  - Out Klassenname
  - print() Methodenname
  - -x bzw. x Parameter

- Zur Herkunft von Out (und der später eingeführten Klasse In)
  - Ausgangspunkt waren die von H. Mössenböck entwickelten Klassen, die für INFOPRMATIK I entsprechend angepasst wurden
  - Die Klassen werden über das IPO bereitgestellt

Information 30: Methode Out.print()

In einem früheren Beispiel kam bereits ein Java-Befehl Out.print(x) zum Einsatz. Dieser Befehl bewirkt, dass der Wert der Variablen x ausgegeben wird. Ein Blick auf die Liste der Schlüsselwörter der Sprache Java macht deutlich, dass weder Out noch print noch die Kombination auftreten.

Wie Information 30 beschreibt, handelt es sich hierbei um eine Java-Klasse Out und die zu dieser Klasse gehörende Methode print() mit dem Aufrufparameter -x bzw. x. Die Klasse stammt aus [Mö03] und wird in den folgenden Programmen zur Ausgabe von Daten genutzt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)


```
class FirstOutProgram {
  public static void main (String[] arg) {
  int v1, v2, max;                          // declare variables
  v1 = 17; v2= -9;                          // assign variables
  max = v1;                                 // determine maximum
  if (max < v2) max = v2;
  Out.print("The maximum of v1 = " + v1 + " and v2 = " + v2 + " is " + max + ".\n");
  }
}
```

- Der an die Methode print() übergebene Parameter setzt sich aus mehreren mittels des Additionssymbols verknüpften Zeichenketten zusammen
  - Zeichenketten (z.B. "The maximum of v1 = ")
  - Integer-Werte (v1, v2, max)

- Ausgabe des Programms

Interakion 31: Ergänzung des Beispiel-Programms um die Ergebnis-Ausgabe

Interakion 31 zeigt, wie das Beispiel-Programm unter Nutzung der Klasse Out zu ergänzen ist, damit das Ergebnis der Berechnung – das Maximum der beiden Variablenwerte – auf dem Bildschirm ausgegeben wird.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
class FirstInOutProgram {
  public static void main (String[] arg) {
  int v1, v2, max;                          // declare variables

  Out.print("\n enter v1: "); v1 = In.readInt();       // read v1 from console
  Out.print("\n enter v2: "); v2 = In.readInt();       // read v2 from console

  max = v1;                                 // determine maximum
  if (max < v2) max = v2;
  Out.println("The maximum of v1 = " + v1 + " and v2 = " + v2 + " is " + max + ".");
  }
}
```

- Neben der Klasse Out zur Ausgabe wird eine Klasse In zur Eingabe über Tastatur oder Datei bereitgestellt

- Methode readInt() liest ganze Zahlen über die Tastatur ein

Information 32: Ergänzung des Beispielprogramms um die Eingabe von Werten

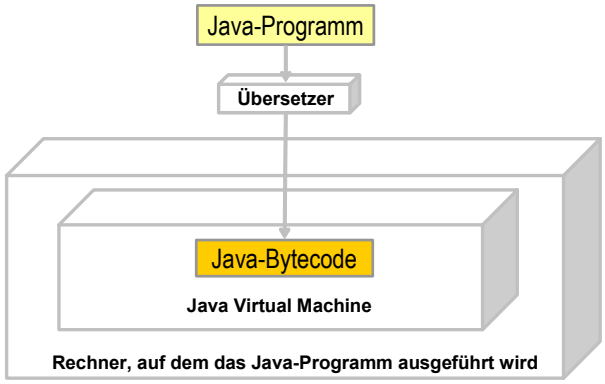
In Information 32 wird das Programm FirstOutProgram zu einem Programm FirstInOutProgram weiterentwickeln, indem die beiden Zahlenwerte v1 und v2 nicht durch eine Zuweisung im Programm sondern durch eine Eingabe über Tastatur bestimmt werden. Hierzu wird die Methode readInt() genutzt, die Bestandteil der Klasse In [Mö03] ist.

3.2.2 Programmübersetzung und -ausführung

Das zuletzt entwickelte Programm (FirstInOutProgram) ist unter der Voraussetzung, dass die Ein- und Ausgabeklassen zur Verfügung stehen, vollständig und soll im Folgenden auf dem Rechner ausgeführt werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Um ein in Java geschriebenes Programm auf einem Rechner ausführen zu können, wird ein spezielles Programm benötigt
  - Übersetzerprogramm, das Java in Java-Bytecode übersetzt
  - der Bytecode wird von einer auf dem Rechner laufenden Java Virtual Machine (JVM) ausgeführt



Information 33: Übersetzung und Ausführung des Java-Programms

Wie die Abbildung in Information 33 zeigt, erzeugt der Übersetzer, der selbst ein umfangreiches ablauffähiges Programm darstellt, aus dem Java-Programm einen so genannten Java-Bytecode. Der Java-Bytecode ist eine maschinennahe Sprache, die von der *Java Virtual Machine* (JVM) ausgeführt wird.

Die JVM heißt deshalb virtuell, weil die Maschine ein in Software realisierter Rechner darstellt, der auf einem realen Rechner ausgeführt wird. Das Konzept der virtuellen Maschine wurde durch Java eingeführt, um die Unabhängigkeit vom Betriebssystem zu erhöhen. Virtuelle Maschinen sind aber nicht zwingend erforderlich, um höhere Programmiersprachen auf einem Rechner ablaufen zu lassen (siehe Sprachen C oder C++).

(0.1) Installieren des Java Development Kit (JDK) mit dem Java-Übersetzer und der Java Virtual Machine

(0.2) Installieren der Klassen In und Out

(1) Erstellen einer Datei
FirstInOutProgram.java
mit dem Java-Quelltext

(2) Übersetzen des Programms mittels
javac FirstInOutProgram.java

(3) Ausführen des Programms mittels
java FirstInOutProgram

Information 34: Konkrete Arbeitsschritte am Beispiel von FirstInOutProgram

Der Java-Übersetzer und die *Java Virtual Machine* sind zunächst auf dem Rechner, auf dem die Java-Programme ausgeführt werden sollen, zu installieren. Diese beiden Programme sind Bestandteil des kostenfrei nutzbaren *Java Development Kit* (JDK). Das JDK sowie die beiden weiter oben eingeführten Klassen In und Out sind auf dem Rechner zu installieren, wobei die jeweiligen Installationshinweise zu beachten sind.

Der erste Schritt der eigentlichen Programmierung besteht dann in der Erstellung des Java-Programms. Hierzu ist auf dem Rechner ein Editor-Programm zu nutzen, mit dem ASCII-Textdateien bearbeitet werden können. Der Name dieser Datei muss mit dem Namen des Programmnamens übereinstimmen, die Endung (*Extension*) muss .java lauten.

Die Übersetzung des in Quelltext vorliegenden Java-Programms in den entsprechenden Bytecode erfolgt durch Aufruf des Befehls javac und der Angabe des Dateinamens und der Endung .java.
Falls der Übersetzungsvorgang erfolgreich abgeschlossen werden konnte, liegt der Bytecode in einer Datei mit dem gleichen Namen wie die Textdatei sowie der Endung .class vor.

Die Ausführung dieses Bytecodes durch die *Java Virtual Machine* erfolgt dann durch den Befehl java und der Angabe des Datei- bzw. Programmnamen ohne Endung.

4 METHODEN

Mit den Methoden wird in diesem Kapitel ein wichtiges Programmierkonzept zur Aufteilung von Anweisungsfolgen eines Programms behandelt [Mö03].

Der Begriff der Methode resultiert aus dem übergeordneten Klassenkonzept, das in der Kurseinheit OBJEKTORIENTIERTE PROGRAMMIERUNG behandelt wird.

- Methoden sind benannte Anweisungsfolgen
  - dienen zur Zerlegung und Modularisierung eines Programms
  - können parametrisiert oder auch parameterlos sein
  - können einen oder keinen Wert zurückliefern
    - werden entsprechend als Funktion oder Prozedur bezeichnet

- Es wurden bereits Methoden verwendet
  - Deklaration der main-Methode: public static void main(String[] arg)
    - Schlüsselwörter "public" und "static" werden später im Zusammenhang mit dem Klassenbegriff geklärt
    - Schlüsselwort "void" besagt, dass die Methode keinen Wert zurückliefert
    - String[] arg ist ein Parameter mit dem Namen arg vom Typ String[]
  - Aufruf der von den Klassen In und Out angebotenen Methoden, z.B.
    - int i = In.readInt();
    - Out.println("Text");

Information 35: METHODEN - Einführung

Methoden traten bereits an verschiedenen Stellen auf (siehe Information 35). Die erste Methode, die eingeführt wurde, war die main()-Methode. Es handelt sich hierbei um eine Methode mit einem Parameter und ohne Rückgabewert. Durch main() wird der Startpunkt eines Java-Programms bestimmt.

Die angegebenen Methoden readInt() und println() der Klassen In und Out sind zwei weitere Beispiele für unterschiedliche Formen von Methoden.

4.1 Deklaration und Aufrufketten

Information 36 zeigt ein Java-Programm mit einer neben der obligatorischen main()-Methodendeklaration weiteren Methodendeklaration printHeader().

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

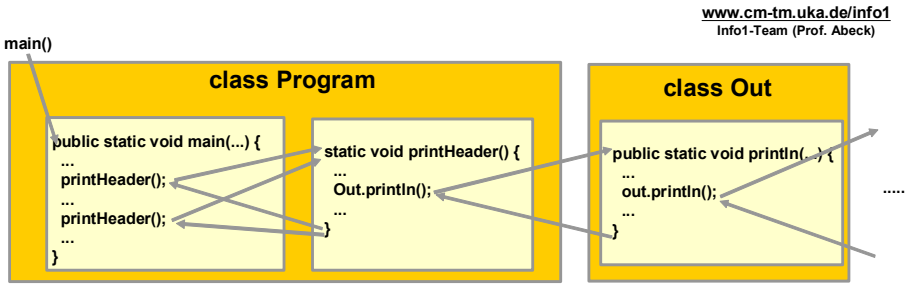
```
class Program {
  static void printHeader() {                  // method declaration: printHeader()
    Out.println("Teilnehmerliste");            // method call: println()
    Out.println("--------------------");        // method call: println()
  }
  public static void main (String[] arg) {     // method declaration: main()
    printHeader();                             // method call: printHeader()
    // names of participants
    printHeader();                             // method call: printHeader()
    // ....
  }
}
```

- Die Methode printHeader()
  - ist wie main() eine Methode zur Klasse Program
  - wird in main() aufgerufen
  - ruft die Methode println() der Klasse Out auf

Information 36: Methodendeklaration und Methodenaufruf

Die Methode printHeader() gehört zur Klasse Program und befindet sich auf der gleichen Ebene wie die main()-Methode.

printHeader() wird von main() 2-mal aufgerufen und ruft seinerseits eine andere Methode – println() aus der Klasse Out – ebenfalls 2-mal auf.



- Namenskonventionen zu Methodennamen
  - beginnen mit einem Kleinbuchstaben
  - beginnen mit einem Verb
  - falls aus mehren Wörtern gebildet, beginnen die Folgewörter mit einem Großbuchstaben

- Namenskonventionen zu Klassennamen
  - beginnen mit einem Großbuchstaben
  - falls aus mehren Wörtern gebildet, beginnen die Folgewörter mit einem Großbuchstaben

Information 37: Methodenaufrufkette und Namenskonventionen

Es entsteht eine Methodenaufrufkette, die graphisch in der Abbildung in Information 37 verdeutlicht wird. Die Kette beginnt mit dem Aufruf der Methode main(). Dieser Aufruf erfolgt

automatisch beim Starten des Programms. Wie in der Abbildung angedeutet wird, ist die Kette nicht vollständig, da in der Methode println() der Klasse Out ein weiterer Methodenaufruf – eine gleichnamige Methode println(), die aber zu einer Klasse out mit klein geschriebenem o gehört – erfolgt.

Die in Information 37 zusammengefassten Namenskonventionen sind aus Gründen der besseren Lesbarkeit eines Programms zwingend einzuhalten. Die Großschreibweise der Klassen wird u.a. auch dadurch motiviert, dass die Sprache Java mit eigenen Klassenbibliotheken arbeitet, für die durchgängig die Kleinschreibweise gewählt wurde. Hierdurch ist somit eine zufällig gleiche Benennung von Klassen ausgeschlossen.

4.2 Parameter

Die Übergabe von Parametern eröffnet erst die zahlreichen Möglichkeiten, die Methoden zur Strukturierung von Programmen bieten.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Parameter sind (Eingabe-) Werte, die eine aufrufende Methode der aufgerufenen Methode übergibt
  - beeinflussen die Ausführung bzw. das Ergebnis der aufgerufenen Methode

```
class PrintMaxProgram {

  static void printMax(int x, int y) {        // x, y: formal method parameters
    if (x > y) Out.println(x); else Out.println(y);
}

  public static void main (String[] arg) {
    printMax(3, 7);                           // 3, 7: parameter values of method call
    int x = In.readInt(); int y = In.readInt();
    printMax(x, y);                           // method call with variables
    printMax(1000 * y, x * y);                // method call with expressions
  } // main
} // class
```

Information 38: Methoden mit Parametern

Wie das Beispiel der in Information 38 programmierten Methode printMax() verdeutlicht, werden die Parameterdeklarationen (hier: int x, int y) geklammert nach dem Methodennamen (hier: printMax()) geschrieben.

Der bei der Methodendeklaration angegebene Methodenkopf bestehend aus Name und Parameterliste bildet die Schnittstelle der Methode, die von einer aufrufenden Methode eingehalten werden muss.

Beim Aufruf einer Methode ist eine Parameterliste anzugeben, der den Parametern, die durch die Schnittstelle vorgegebenen sind, gerecht wird. Die Schnittstellen-Parameter einer Methode werden auch als formale Parameter bezeichnet, während die Aufrufparameter aktuelle Parameter genannt werden.

Wie das Programm zeigt, können die beim Aufruf der Methode printMax() übergebenen aktuellen Parameter einfache ganzzahlige Werte, Variablen oder Ausdrücke sein.

- Übergabe der aktuellen Parameter an die formalen Parameter
  - ggf. Berechnung der Ausdrücke der aktuellen Parameter
  - es handelt sich um Zuweisungen, weshalb Typkompatibilität gefordert ist

- Die Art der Übergabe wird als Call-by-Value bezeichnet
  - formaler Parameter kopiert den Wert des aktuellen Parameters in eine getrennte Speicherzelle
  - Änderungen, die ggf. am formalen Parameter im Methodenrumpf vorgenommen werden, beeinflussen den Wert des aktuellen Parameters nicht

Information 39: Parameterübergabe

Im Zusammenhang mit der Parameterübergabe sind eventuelle Ausdrücke, die als aktuellen Parameter übergeben werden, zunächst zu berechnen. Wichtig ist dabei die Einhaltung der Typkompatibilität (siehe Information 39).

Die Parameterübergabe erfolgt nach dem *Call-by-Value*-Prinzip. Alternative Übergabekonzepte sind das *Call-by-Name*-Prinzip bzw. das *Call-by-Reference*-Prinzip, die eine Änderung des aktuellen Parameterwertes gemäß den Änderungen des formalen Parameters bewirken.

Von Java wird für Eingabeparameter von einfachem Typ (z.B. `int`, `boolean`) ausschließlich das *Call-by-Value*-Prinzip unterstützt.

4.3 Funktionen

Bislang wurden Methoden ohne Rückgabewerte behandelt. Solche Methoden werden auch als Prozeduren bezeichnet. Liefert eine Methode einen Rückgabewert an die aufrufende Methode, handelt es sich um eine Funktion.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
class FctMaxProgram {
  static int fctMax(int x, int y) {          // function which provides int result
    if (x > y) return x; else return y;      // return statement
  }

  public static void main (String[] arg) {
    int x = 2;
    Out.println(fctMax(x, 5 * x) * 100);     // function call in parameter expression
                                             // prints: _____
  }
}
```

- Rückgabe des Funktionsergebnisses mittels return-Anweisung
- Funktionen werden üblicherweise als Operanden in einem Ausdruck verwendet
- Es kann immer nur ein Ergebniswert zurückgeliefert werden
  - mehrere Ergebniswerte sind in Form eines Objekts zusammen zu fassen

Interaktion 40: Funktionen

Es bietet sich an, die Prozedur printMax() in eine entsprechende Funktion fctMax() zu überführen, die das Maximum nicht ausdruckt sondern als Ergebniswert an die aufrufende Methode übergibt. Hierdurch ergibt sich unmittelbar der Vorteil, dass das von einer Funktion gelieferte Ergebnis als Operand in einem komplexeren Ausdruck genutzt werden kann, wie das Programmbeispiel in Interaktion 40 zeigt.

Zur Rückgabe des Funktionsergebnisses dient die return-Anweisung. Eine Funktion muss durch eine solche return-Anweisung zwingend abgeschlossen werden, was auch durch den Übersetzer überprüft wird.

Mittels einer return-Anweisung kann immer nur ein Ergebniswert zurückgegeben werden. Besteht das Ergebnis aus mehreren Einzelergebnissen, sind (zusammengesetzte) Objekte einzuführen, wie in einer späteren Kurseinheit verdeutlicht wird.

Es sei angemerkt, dass die return-Anweisung auch bei Prozeduren eingesetzt werden kann. Hier wirkt diese ohne Parameter ausgeführte Anweisung wie die break-Anweisung, d.h. die Prozedur wird abgeschlossen und die Kontrolle geht an die aufrufende Methode über.

4.4 Lokale und globale Namen

Die bislang behandelten Methoden haben ausschließlich Anweisungen enthalten. Daneben können auch Deklarationen von Variablen und Konstanten auftreten.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- In einer Methode können Variablen und Konstanten (aber keine weiteren Methoden) deklariert sein
  - sind wie die formalen Parameter lokal zu dieser Methode deklariert, d.h. nur in dieser Methode gültig

```
static void m (int x) {
  final boolean debug = true;
  int y;
  float z;
  ...
  if (debug) Out.println(...);
  ...
}
```

- Methode m() deklariert drei lokale Variablen x, y, z und eine lokale Konstante debug
  - dürfen nur in m() verwendet werden
  - sind außerhalb m() nicht sichtbar
  - "leben" nur während der Ausführung der Methode m()

Information 41: Lokale Variablen und Konstanten

Die in einer Methode deklarierten Variablen und Konstanten sind wie die formalen Parameter lokale Namen der Methode. "Lokaler Name einer Methode" heißt, dass der Name nur in solchen Anweisungen genutzt werden darf, die Teil der Methode sind.

Bezogen auf das in Information 41 angegebene Beispiel bedeutet diese Festlegung, dass die Variablen x, y, z und die Konstante debug nur lokal innerhalb der Methode m() verwendet werden dürfen. Es handelt sich um so genannte lokale Variablen und Konstanten.

Bei Aufruf der Methode m() wird für die lokalen Größen x, y, z und debug der benötigte Speicherplatz bereitgestellt und am Ende der Methodenausführung wird dieser Speicherplatz wieder frei gegeben.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
class Program {
  static int a; static float b;      // declarations of global variables
  static final int c = 3;            // declaration of global constant

  static void m (int x) {
    int y; float z;                  // x, y, z: local
    ... // which variables/constants can be used here? _____
  }

  public static void main (String[] arg) {
    ... // which variables/constants can be used here? _____
  } /* main */   } /* class */
```

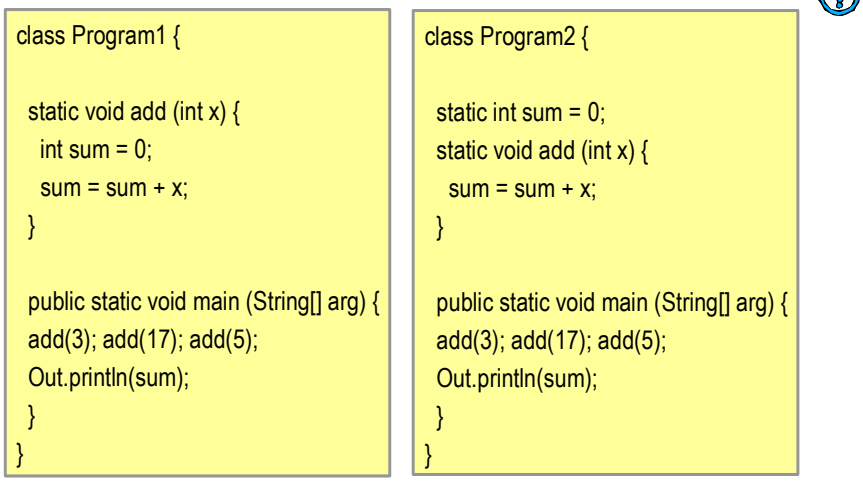
- Globale Variablen und Konstanten sind außerhalb einer Methode in der Klasse deklariert
  - sind mit static deklariert
  - können in allen Methoden der Klasse benutzt werden

Interaktion 42: Globale Variablen und Konstanten

Jede Methode ist in einer Klasse deklariert. Auf Klassenebene können neben Methoden auch Variablen und Konstanten deklariert werden, die dann in allen Methoden dieser Klasse benutzt werden können. Solche außerhalb der Methoden deklarierten Variablen und Konstanten heißen globale Variablen.

Am Beispiel des Programms in Interaktion 42 ist zu klären, welche lokalen bzw. globalen Variablen an den beiden gekennzeichneten Stellen im Programm genutzt werden können.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)



```
class Program1 {

  static void add (int x) {
    int sum = 0;
    sum = sum + x;
  }

  public static void main (String[] arg) {
  add(3); add(17); add(5);
  Out.println(sum);
  }
}
```

```
class Program2 {

  static int sum = 0;
  static void add (int x) {
    sum = sum + x;
  }

  public static void main (String[] arg) {
  add(3); add(17); add(5);
  Out.println(sum);
  }
}
```

- Welche Ausgabe liefern die Programme?

  Program1 _____ Program2 _____

Interaktion 43: Lokale und globale Variable sum

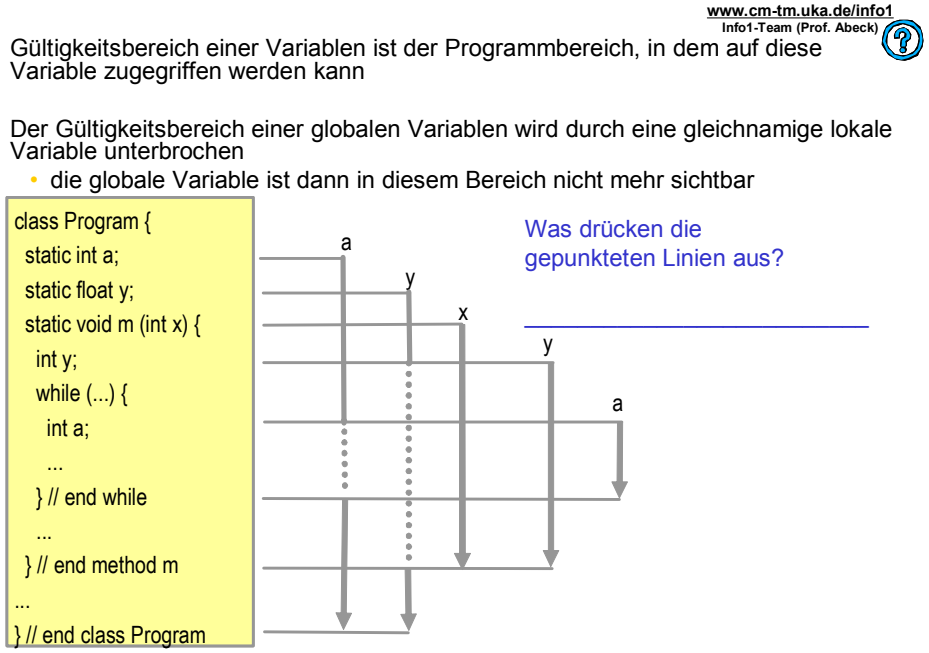
Das in Interaktion 43 behandelte Beispiel hat offensichtlich zum Ziel, die Summe einer Zahlenfolge zu bilden. Während Program1 die Aufgabe durch die Verwendung einer lokalen Variablen sum zu lösen versucht, verwendet Program2 eine globale Variable sum.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Grundsätzlich gilt: lokal vor global
- Wenn möglich, sind Variablen lokal und nicht global zu deklarieren
  - globale Variablen nur in begründeten Fällen verwenden
- Vorteile lokaler Variablen gegenüber globalen Variablen
  - eine Methode lässt sich verstehen, ohne das Umfeld kennen zu müssen
  - eine Methode manipuliert bei schreibendem Zugriff auf globale Variablen in schwer nachvollziehbarer Form das Umfeld (Seiteneffekte)
  - Problem der Namenskonflikte zwischen lokalen und globalen Variablen besteht nicht

Information 44: Verwendung lokaler oder globaler Variablen

Globale Variablen existieren im Gegensatz zu lokalen Variablen über Methodengrenzen hinweg. Daher sind Änderungen von globalen Variablen, die in einer Methode erfolgen, auch außerhalb der Methode wirksam. Eine Methode kann somit Ergebnisse über globale Variablen als so genannte "Seiteneffekt" nach außen weitergeben. Diese Form der "impliziten" Ergebnisweitergabe von Methoden sollte nur in ganz bestimmten Fällen genutzt werden, da hierdurch die Wirkungsweise einer Methode undurchsichtig und die Methoden-Schnittstelle "aufgeweicht" wird. Information 44 fasst die Vorteile von lokalen gegenüber den globalen Variablen zusammen.



Information 45: Gültigkeitsbereich (Sichtbarkeitsbereich) einer Variablen

Wie der Programmausschnitt in Information 45 verdeutlicht, können lokale und globale Variablen (zufällig) den gleichen Namen haben. Es stellt sich die Frage, auf welche Variable im Falle der Existenz einer gleichnamigen globalen und lokalen Variablen zugegriffen wird. Die Antwort führt über den Gültigkeitsbereich, der auch als Sichtbarkeitsbereich bezeichnet wird. Wie die gepunktete Linie andeutet, werden globale Variablen durch lokale Variablen verschattet, d.h. die lokale Variable ist gültig und wird zugegriffen. Konkret gilt also innerhalb der Methode m() die deklarierte lokale Variable int y und nicht die in der Klasse Program deklarierte globale Variable static float y. Wie die durchgezogene Pfeillinie andeutet, wird der Gültigkeitsbereich der globalen Variablen float y nach Verlassen der Methode m() fortgesetzt, weil hiermit der Gültigkeitsbereich der lokalen Variablen int y endet.

- Das Zeitintervall, in dem eine Variable einen Speicherplatz beansprucht, wird als Lebensdauer der Variablen bezeichnet
- Lokale und globale Variablen werden in verschiedenen Speicherbereichen angelegt
  - lokale Variablen im (Laufzeit-) Keller (stack)
  - globale Variablen auf der Halde (heap)
- Der Keller wächst bzw. schrumpft mit jedem Methodenaufruf bzw. mit jedem Verlassen einer Methode
  - im Keller sind zu jedem Zeitpunkt die lokalen Variablen und Konstanten der aktuellen Methodenaufrufkette gespeichert

Information 46: Lebensdauer und Speicherorganisation

Während der Gültigkeitsbereich festlegt, wann eine Variable benutzt werden kann, macht die Lebensdauer eine Aussage darüber, wie lange die Variable einen Platz im Speicher belegt. Offensichtlich gibt die globale Variable im obigen Beispiel während der Verschattung durch die lokale Variable ihren Speicherplatz nicht frei. Allerdings ist ein Zugriff auf die Speicherzellen in diesem Programmbereich nicht möglich.

Wie Information 46 beschreibt, werden globale und lokale Variablen in unterschiedlichen Speicherbereichen gehalten. Ein Charakteristikum des als Keller (*Stack*) bezeichneten Speicherbereichs, in dem die lokalen Variablen gehalten werden, ist dessen Anwachsen und Schrumpfen. Der Keller ist eine wichtige Datenstruktur in der Informatik.

5 DATENTYPEN

Die einzigen Datentypen, die in den bisher behandelten Programmen bzw. Programmausschnitten vorkommen, sind boolesche Werte und ganze Zahlen. Zur Darstellung ganzer Zahlen stehen – je nach dem benötigten Wertebereich – die in Information 47 angegebenen Datentypen in Java zur Verfügung [Mö03].

- Bislang wurden neben dem Datentyp boolean ausschließlich Datentypen zur Darstellung ganzer Zahlen benutzt
  - Datentypen byte, short, int, long
- Neben den ganzen Zahlen unterstützt Java auch das Rechnen mit Gleitkommazahlen
  - Datentypen float, double
- Eine weitere elementare Form von Daten sind die Zeichen
  - Datentyp char

Information 47: WEITERE DATENTYPEN - Überblick

In diesem Kapitel werden mit den Gleitkommazahlen und den Zeichen zwei weitere elementare Datentypen eingeführt [Mö03].

5.1 Gleitkommazahlen

Gleitkommazahlen werden in Form der in Information 48 dargestellte Exponentendarstellung geschrieben.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Gleitkommaformat
  - Syntax: Kommazahl 'E' GanzeZahl
  - Wert: Kommazahl * $10^{GanzeZahl}$
  - Beispiele
    - 0.0314E2 Wert: 0.0314 * 10^2 = 3.14
    - 999E-3 Wert: 999 * 10^{-3} = 0.999

- Datentypen in Java zur Darstellung von Gleitkommazahlen
  - float 32 Bits ≈ 1.4E-45 ... ≈ 3.4E38
  - double 64 Bits ≈ 4.9E-324 ... ≈ 1.8E308

```
float x,y;  // declaration of two float variables x and y
double z; // declaration of one double variable z
```

Information 48: Gleitkommazahlen

Die ganze Zahl nach dem Buchstaben 'E' (auch die Kleinschreibweise 'e' ist zulässig) gibt die Zehnerpotenz an, mit der die Kommazahl multipliziert wird.

Es werden die zwei Datentypen float und double zum Arbeiten mit Gleitkommazahlen angeboten. Der Datentyp double ist dann zu nutzen, wenn der durch den Datentyp float angebotene Wertebereich nicht ausreicht.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Harmonische Reihe bis zum n. Glied:
  - 1/1 + 1/2 + 1/3 + ... + 1/n

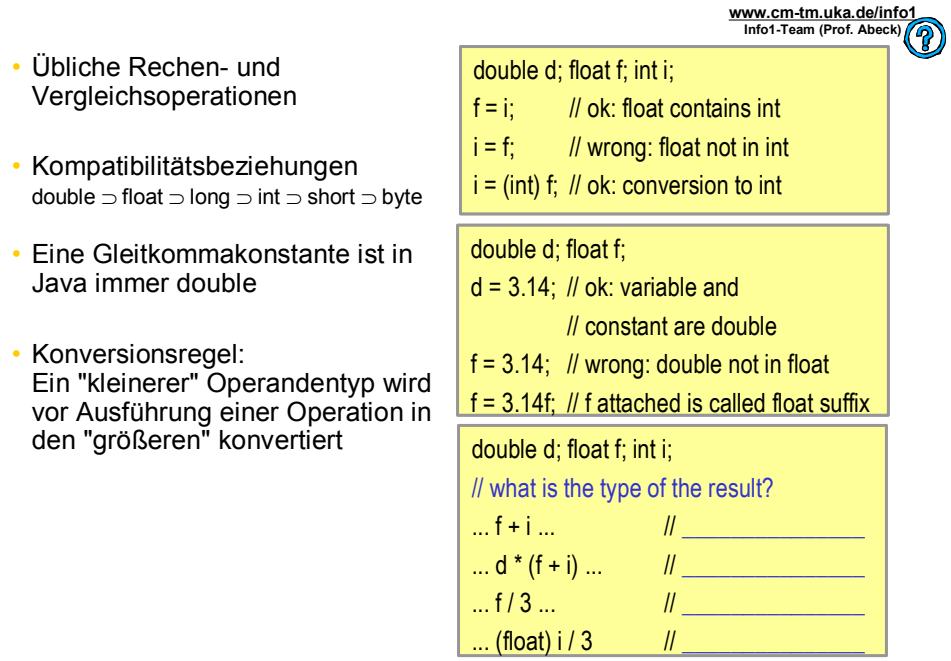
- Berechnung in einem Java-Programm mittels
  1. Einlesen der ganzen Zahl n
  2. Deklaration einer float-Variablen sum, die zur Aufnahme des Ergebnisses dient
  3. Berechnung der harmonischen Reihe in einer while-Schleife
     - hier absteigend, d.h. 1/n + 1/(n-1) + 1/(n-2) + ... + 1/2 + 1
  4. Ausgabe des Ergebnisses

```
int n = _____________________________;          // 1
_____________________;                          // 2
int i = n;                                      // 3
while (i > 0) {                                 // 3
  sum = sum + (float) _________________;        // type casting
  i = i - 1;                                    // 3
}                                               // 3
_____________________________________;          // 4
```

Interaktion 49: Berechnung der Harmonischen Reihe

In Interaktion 49 wird anhand einer einfachen Programmieraufgabe, der Berechnung der Harmonischen Reihe, das Arbeiten mit Gleitkommazahlen aufgezeigt. Das Programm ist mit Hilfe der angegebenen Beschreibung entsprechend zu vervollständigen.

Auf das Rechnen mit Gleitkommazahlen und Typkonversionen wird in der folgenden Interaktion 50 näher eingegangen.



Interaktion 50: Eigenschaften der Datentypen double und float

Mit Gleitpunktkommazahlen kann ähnlich wie mit den ganzen Zahlen gerechnet werden. An Rechenoperationen stehen die Addition (+), die Subtraktion (-), die Multiplikation (*), die Division (/) und die Modulo-Operation (%) zur Verfügung. Dabei ist die Modulo-Operation so definiert, dass für zwei Gleitkommazahlen x und y die Operation x % y eine Gleitkommazahl r ergibt, die den Rest der Division von x und y darstellt (also r = x – q * y, wobei q der ganzzahlige Teil von x / y ist).

Interaktion 50 zeigt die Kompatibilitätsbeziehung der beiden Datentypen double und float sowie der bereits eingeführten Datentypen zu den ganzen Zahlen auf. Eine Zuweisung eines umfassenden Typs ist nur nach vorheriger Typkonversion möglich, wobei zu beachten ist, dass hierdurch ein Informationsverlust entstehen kann.

Während Java ganzzahligen Konstanten bekanntlich den Datentyp int zuordnet, sind Gleitkomma-Konstanten immer automatisch vom Datentyp double. Daher muss bei einer Zuweisung einer Gleitkomma-Konstanten zu einer float-Variablen an die Konstante ein f angehängt werden, damit dieses nicht vom Typ double sondern vom Typ float ist.

Aufgrund des erheblich höheren Rechenaufwands, der durch die Gleitkommaoperationen entsteht, sollten die Datentypen double und float auch wirklich nur dann genutzt werden, wenn die Nachkommastellen erforderlich sind.

5.2 Zeichen

Neben den Zahlen gehören die Zeichen zu den wichtigsten Daten, die in Programmen verarbeitet werden müssen. In Java steht für den Umgang mit Zeichen der Datentyp char zur Verfügung. Wie bei Zahlen können Zeichen als Konstanten und Variablen auftreten. Zunächst werden die Zeichen-Konstanten näher betrachtet.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Zeichen sind wie Zahlen Daten von einem besonderen Typ
  - der Typ trägt in Java die Bezeichnung char (steht für character)
  - wie bei Zahlen lassen sich Zeichen-Konstanten und Zeichen-Variablen unterscheiden
- Zeichen-Konstanten
  - werden in Hochkommata (') gestellt
    - Beispiele: 'a' 'b' 'A' '?'
  - werden durch einen Zeichencode in Zahlen umgewandelt
  - bekannte Zeichencodes sind ASCII (American Standard Code of Information Interchange) und Unicode

Information 51: Zeichen

Information 51 zeigt Beispiele von Konstanten, die sich durch einem zugrunde liegenden Zeichencode in Zahlenwerte umwandeln lassen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- ASCII definiert 128 Zeichen
- Ein ASCII-Zeichen wird durch 1 Byte dargestellt
- Zeichenbereich 0x00 bis 0x1f und Zeichen 0x7f sind nicht sichtbare Zeichen, wie z.B.

0x0a	neue Zeile	linefeed (LF)
0x0d	Zeilenende	carriage return (CR)
0x09	Tabulatorsprung	horizontal tab (HT)
0x7f	Löschzeichen	delete (DEL)

- Zeichenbereich 0x20 bis 0x7e enthält druckbare Zeichen, wie z.B.

0x20	Leerzeichen
0x30 - 0x39	Zahlbereich 0 bis 9
0x41 - 0x5a	Großbuchstaben 'A' bis 'Z'
0x61 - 0x7a	Kleinbuchstaben 'a' bis 'z'

Information 52: Zeichencode ASCII

Der in Information 52 näher beschriebene Zeichencode ASCII – *American Standard Code of Information Interchange* – ist ein 1-Byte-Code, durch den insgesamt 128 Zeichen definiert sind.

Die Organisation des Zeichenbereichs in nicht sichtbare und druckbare Zeichen ist in Information 52 ausgeführt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Im ASCII-Zeichensatz fehlen verschiedene Zeichen, die in bestimmten Problembereichen zwingend erforderlich sind
  - z.B. mathematische Zeichen, griechische Symbole, Umlaute, arabische, chinesische, ... Zeichen

- In Java wird daher der den ASCII-Zeichensatz erweiternde Zeichensatz Unicode verwendet
  - 2-Byte-Code (d.h 2^{16} = 65 536 mögliche Zeichen)
  - wird als 4-stellige Hexadezimalzahl \unnnn beschrieben
  - z.B. \u000a (LF), \u0009 (HT), \u00e4 (ä), \u03b1 (α)

- Für häufig vorkommende Steuerzeichen können speziell von Java vorgesehene Escape-Sequenzen benutzt werden
  - z.B. '\n' (\u000a), '\\' (backslash \u005c), '\'' (single quote \u0027)

Information 53: Zeichencode Unicode

Die 128 durch den ASCII-Zeichencode festgelegten Zeichen reichen in Bereichen, in denen z.B. mathematische Zeichen oder Zeichen gewisser Sprachen (z.B. griechisch, kyrillisch, arabisch oder Umlaute der deutschen Sprache) benötigt werden, nicht aus. Daher wird in Java der so genannte Unicode-Zeichensatz verwendet, der die ASCII-Zeichen (\u0000 bis \u007f, zur Schreibweise siehe Information 53) übernimmt und diese ergänzt:

Durch die Bereitstellung von *Escape*-Sequenzen – also Zeichenreihen, in denen der als *Escape*-Zeichen genutzte Backslash '\' verwendet wird – lassen sich die häufig verwendeten Steuerzeichen in einer übersichtlicheren Form innerhalb von Programmen nutzen.

Variablen, die anstelle von Zahlen die oben beschriebenen Unicode-Zeichen als Werte tragen sollen, werden in Java mit dem Datentyp char deklariert.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Zeichenvariablen werden mit dem Typ char deklariert

- Zeichen werden intern durch den Unicode in Zahlenwerte codiert
  - mit char-Werten lässt sich "rechnen"

- Typ char wird in der Hierarchie der Standardtypen auf der gleichen Stufe wie short eingeordnet

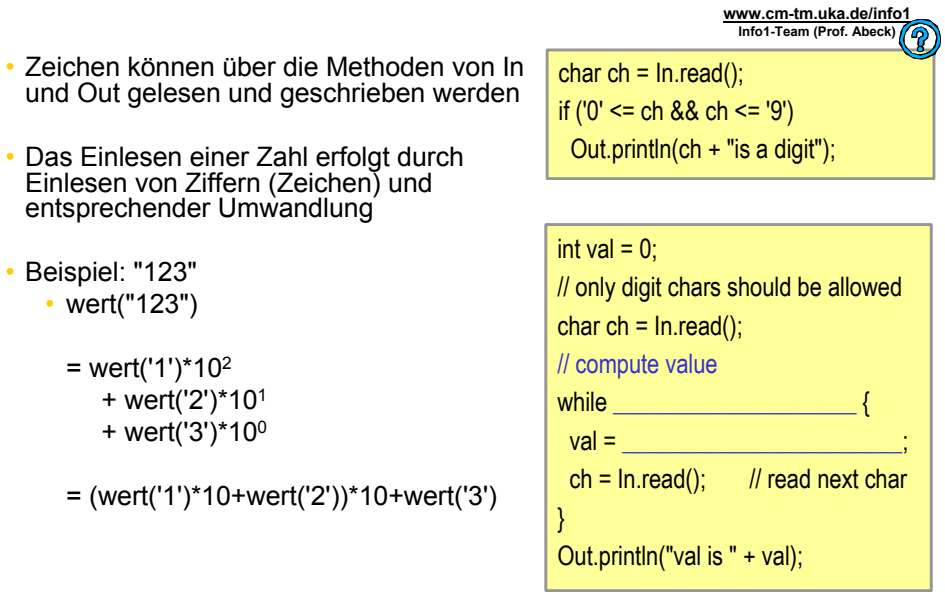
double ⊃ float ⊃ long ⊃ int ⊃ short ⊃ byte
⊃ char

```
char ch1, ch2 = 'c';
ch1 = 'a';

...

int i = ch1 - ch2;      // i == _____
...
ch1 = ch2;              // ok? _____
i = ch1;                // ok? _____
                        // i == _____
ch2 = (char) (i + 1); // ok? _____
                        // ch2 == ___
```

Interaktion 54: Zeichenvariablen

Einer Zeichenvariablen kann bereits bei der Deklaration oder in einer separaten Zuweisung ein Zeichenwert zugewiesen werden. Intern wird bei der Deklaration einer Zeichenvariablen ein 2-Byte großer Speicherplatz vorgesehen, in dem bei Zuweisung eines Zeichenwertes zu dieser Variablen der entsprechende Unicode-Zahlenwert gespeichert werden.
Aufgrund der internen Darstellung von Zeichenvariablen lässt sich hiermit "rechnen" wie mit Zahlvariablen.

In der in Interaktion 54 gezeigten Standardtypen-Hierarchie ist der Datentyp char wie short eingeordnet, weshalb z.B. die Zuweisung des Wertes einer Zeichenvariablen zu einer int-Variablen ohne Typ-Konversion zulässig ist. Im umgekehrten Fall – also im Falle einer Zuweisung des Wertes einer int-Variablen zu einer Zeichenvariablen – ist eine Typ-Konversion erforderlich.



Interaktion 55: Zeichen und Ein-/Ausgabe

Zeichen müssen sich auch in ein Programm einlesen und von einem Programm ausgeben lassen. Hierzu stellen die weiter oben eingeführten Klassen In und Out entsprechende Methoden zur Verfügung.

Wie der zweite Programmausschnitt zeigt, baut die Eingabe einer Zahl auf der Zeicheneingabe auf: Die eingegebene Zahl ist für das Programm zunächst eine Folge von Zeichen. Jedes der Zeichen ist in der Menge {'0', ..., '9'} enthalten. Die eigentliche Zahl wird durch das in Interaktion 55 angegebene Vorgehen, das in eine entsprechende Schleife umzusetzen ist, ermittelt. Die Umformung der Berechnung in die geklammerte Form wird als Horner-Schema bezeichnet.

- Die Java-Bibliothek bietet zahlreiche Operationen im Zusammenhang mit Zeichen an
  - Verwendung von Bibliotheksfunktionen (Standardfunktionen) ist Teil eines guten Programmierstils
  - erhöht die Stabilität und vermindert die Komplexität von Programmen

- Beispiele
  - Character.isLetter(ch)
    - liefert true, falls ch ein Buchstabe ist
  - Character.isDigit(ch)
    - liefert true, falls ch eine Ziffer ist
  - Character.toUpperCase(ch)
    - liefert den entsprechenden Großbuchstaben, falls ch ein Kleinbuchstabe ist

Information 56: Bibliotheksfunktionen

Da verschiedene Zeichenoperationen – wie z.B. die im obigen Programm verwendete Funktion zur Überprüfung, ob ein Zeichen eine Ziffer ist – immer wieder auftreten, stellt Java eine umfangreiche Bibliothek dieser Operationen als so genannte Standardfunktionen bereit.

In dieser Veranstaltung werden solche nahe liegenden Operationen als Beispiele verwendet, um die Programmierung im Kleinen zu üben. Daher werden diese Standardfunktionen – wie im obigen Beispiel die Standardfunktion isDigit(ch) – ausnahmsweise nicht verwendet.

Hiermit sind die Grundlagen der Programmierung, die das Erstellen erster vollständiger Java-Programme ermöglichen, abgeschlossen. Die vorgestellten Programmierelemente werden durch die Kurseinheiten zur imperativen und objektorientierten Programmierung [C&M-IP, C&M-OP] sukzessive ergänzt.

VERZEICHNISSE

Abkürzungen und Glossar

Abkürzung oder Begriff	**Langbezeichnung und/oder Begriffserklärung**
ASCII	*American Standard Code of Information Interchange* Weit verbreiteter 1-Byte-Code, durch den Zeichen in 7 Bit langen Binärzahlen codiert werden (achtes Bit ist ungenutzt). Ein den ASCII-Code erweiternder Code ist der so genannte Unicode.
BNF	Backus-Naur-Form Schreibweise für Regeln einer Grammatik.
Call-by-Value-Prinzip	Prinzip im Zusammenhang mit der Parameterübergabe, das besagt, dass die Werte der Aufrufparameter (aktuelle Parameter) an die entsprechenden formalen Parameter übergeben werden. Insbesondere werden die Werte der aktuellen Parameter durch den Aufruf nicht verändert.
EBNF	Erweiterte BNF Erweiterung der BNF um Metaregeln zur ausdrucksstärkeren Formulierung von Regeln einer Grammatik.
Escape-Zeichen	Ein ausgezeichnetes Zeichen, das zur Darstellung spezieller Sequenzen (*Escape*-Sequenzen) dient. Beispiel: In Java dient das Zeichen '\' (*Backslash*) als *Escape*-Zeichen, um beispielsweise mittels '\n' einen Zeilenumbruch darzustellen.
Funktion	Im Kontext der Java-Programmierung ist eine Funktion eine Methode, die einen Rückgabewert an die aufrufende Methode liefert.
Grammatik	Regelsystem, durch das die Syntax einer (formalen) Sprache, z.B. einer Programmiersprache, festgelegt wird.
JDK	*Java Development Kit* Softwarepaket, das einen Java-Übersetzer und die JVM beinhaltet.
JVM	*Java Virtual Machine* Software, die den Java-Bytecode ausführt. Eine virtuelle Maschine bezeichnet einen in Software realisierten Prozessor.
Methode	Bezeichnung der in Klassen auftretenden Rechenvorschriften, denen Parameter übergeben werden können und die einen Rückgabewert abliefern können. Namenskonvention: Methodennamen beginnen mit einem kleingeschriebenen Buchstaben. Beispiel: Methode `main()` ist eine in Java ausgezeichnete Methode, da diese den Einstiegspunkt in das Programm markiert.

Programmieren	Beschreibung eines Problems in einer Form, dass es mittels eines Rechensystems gelöst werden kann.
Schleife	Sprachelement einer Programmiersprache, durch das eine mehrmalige Ausführung von Programmteilen ermöglicht wird.
Schlüsselwörter	Wörter einer Sprache, die deren grundlegende Syntax angibt. Die Sprache Java besteht beispielsweise aus 48 Schlüsselwörtern. Beispiele: if, for, class, void
Software-Entwicklung	Prozess der Erstellung eines Programms. Der Prozess zerfällt in mehrere iterativ durchlaufene Phasen. Verwandte Begriffe: *Software Engineering*, Programmieren (im Großen)
UML	*Unified Modeling Language* Sprache, die aus graphischen Elementen zur semi-formalen Beschreibung von beliebigen Gegenständen (z.B. Software-Systeme oder Geschäftsbereiche) besteht.
von-Neumannscher Flaschenhals	Bezeichnung eines Engpasses, der in der von-Neumann-Architektur aus der im Vergleich zur Prozessorbearbeitungszeit relativ langen Speicherzugriffszeiten entsteht.
Zuweisung	Wichtige Form einer Anweisung, durch die einer Variablen (linke Seite) ein Wert (rechte Seite) zugewiesen werden kann.
Zuweisungskompatibilität	Bei einer Zuweisung muss der Typ der rechten Seite den Typ der linken Seite einschließen.

Index

Informationen und Interaktionen

Literatur

[C&M-SM] Cooperation&Management, SYSTEMMODELLIERUNG, Kursdokument zur Vorlesung "INTERNET-SYSTEME UND WEB-APPLIKATIONEN",

http://www.cm-tm.uka.de/iswa, Universität Karlsruhe (TH), C&M (Prof. Abeck).

[Me90] Bertrand Meyer: Objektorientierte Softwareentwicklung, Carl Hanser Verlag, 1990.

[Mö03] Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

[Oe01] Bernd Oestereich: Objektorientierte Software-Entwicklung – Analyse und Design mit der Unified Modeling Language, Oldenbourg Verlag, 2001.

Kursblock

FORMALE GRUNDLAGEN

Dieser Kursblock behandelt die zentralen Themen der theoretischen Informatik. Zunächst werden die wichtigsten Grundbegriffe, auf denen die Informatiktheorie aufbaut, eingeführt. Anschließend werden die algebraischen Strukturen und formalen Systeme beschrieben, auf denen die Kurseinheit zu den Rechenstrukturen und funktionalen Programmen aufsetzt.

Die drei Kurseinheiten dieses Kursblocks sind:

GRUNDBEGRIFFE DER INFORMATIK

Kurzbeschreibung

Die Kurseinheit führt in zentrale Grundbegriffe der Informatik (Information, Algorithmus, Modell, Architektur) ein und vertieft diese anhand konkreter Beispiele.

Schlüsselwörter

Information, Signal, Wissen, Semiotik, Algorithmus, Euklidischer Algorithmus, Modell, Wirklichkeit, objektorientierte Modellierung, Rechnerentwicklung, von-Neumann-Architektur, Software-Architektur

Lernziele

1. Die Beziehungen, die der Begriff der Information zu Signal, Nachricht, Syntax, Semantik, Wissen und Codierung aufweist, sind bekannt.
2. Die wesentlichen Eigenschaften eines Algorithmus werden verstanden.
3. Das Modell als Abbildung eines Ausschnitts der Wirklichkeit und die Bedeutung des Modellierens können nachvollzogen werden.
4. Die den Systemen zugrunde liegenden Architekturprinzipien können wiedergegeben werden.

Hauptquellen

- Gerhard Goos: Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer Verlag 1997.

Inhaltsverzeichnis

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- INFORMATION
  - Signal, Information, Wissen, Syntax, Semantik, Codierung
- ALGORITHMUS
  - Definition, Ablaufelemente, Beispiel, Effizienz
- MODELL
  - Wirklichkeit und Modell, objektorientierte Modellierung
- ARCHITEKTUR
  - von-Neumann-Rechner, Softwarearchitektur

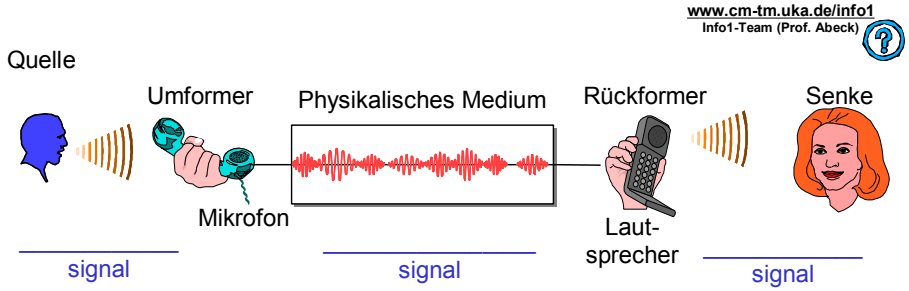
Information 1: GRUNDBEGRIFFE DER INFORMATIK

1 INFORMATION

Das Kunstwort Informatik ist durch den Begriff der Information (sowie durch den Begriff der Mathematik) geprägt. In diesem Kapitel werden die wichtigsten Begriffe und Sachverhalte, die mit dem Begriff der Information in Zusammenhang stehen, skizziert [Go97].

1.1 Signal

Den Ausgangspunkt der Überlegungen, die zur Information führen, bildet das Signal [C&M-BS].



- Signal als Ausgangspunkt für den Informationsbegriff
  - Signalparameter sind die Eigenschaften des Signals, die der zeitlichen Veränderung unterliegen
  - Signalübertragung ist die Weitergabe von Mitteilungen auf der Ebene der Signale
- Inschrift oder schriftliche Darstellung ist die dauerhafte Darstellung einer Mitteilung auf einem physikalischen Medium

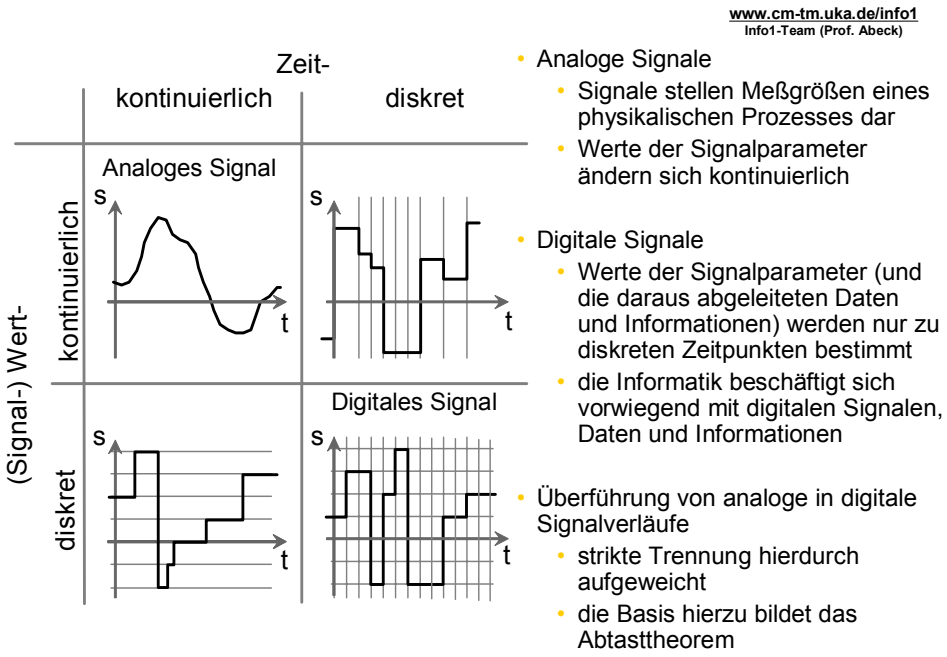
Interaktion 2: INFORMATION – Signal und Inschrift

Eine gesprochene Mitteilung wird in Form von Schallwellen ausgedrückt. Falls mit dem Empfänger, der die Mitteilung erhalten hat, über das Telefon kommuniziert wurde, wie in der Abbildung in Interaktion 2 dargestellt ist, hat eine zusätzliche Transformation in elektromagnetische Impulse oder Lichtwellen stattgefunden, die beim Empfänger wieder

umgesetzt wurden und als Schallwellen aus dem Hörer kommen. In diesem Fall wird ein Übertragungssystem bestehend aus einem Umformer und einem Rückformer genutzt, die eine Transformation vom Primärsignal in das Übertragungssignal und wieder zurück durchführen.
Die beiden Signaltypen, die im Zusammenhang mit dem Umformer und dem Rückformer unterschieden werden, sind in Interaktion 2 zu ergänzen.

Häufig besteht die Anforderung, eine mittels Signalen gebildete Mitteilung dauerhaft (persistent) zu speichern, woraus die so genannte Inschrift resultiert. Die Medien zur Speicherung von Inschriften werden als Schriftmedien bezeichnet.

Eine nähere Analyse der Signale führt zu den in Information 3 [C&M-BS] angegebenen Signalklassen.



Information 3: Signalklassen

Die analogen Signale resultieren aus den zeitlich sich kontinuierlich ändernden physikalischen Prozessen. Ein Beispiel eines physikalischen Prozesses ist die Beschleunigung eines Fahrzeugs, bei dem die Geschwindigkeit kontinuierlich mit der Zeit zunimmt. Digitale Signale entstehen dadurch, dass die Werte der Signalparameter nur zu diskreten Zeitpunkten bestimmt werden.

Die Signalklassen werden durch den Signalverlauf bestimmt, d.h. durch die Änderung des Signalwerts s über der Zeit t. Dabei gilt sowohl für den Signalwert als auch für den Zeitverlauf die Unterscheidung von kontinuierlich (stetig) und diskret (sprunghafter Verlauf beschreibt, bei dem nur bestimmte Werte gültig sind). Diese Unterscheidung ergibt die vier in Information 3 angegebenen Signalklassen.

Es stellt sich die Frage, ob sich analoge und digitale Signale geeignet ineinander überführen lassen. Die Vermutung ist zunächst, dass durch die Wahl der diskreten Zeitpunkte beim Übergang „von analog zu digital“ Information verloren geht. Falls diese diskreten Zeitpunkte genügend kurz nacheinander gewählt werden, entsteht kein Informationsverlust. Diese Aussage wird durch das Abtasttheorem präzisiert.

Im Bereich der Telekommunikation wurden Verfahren entwickelt, um ein Signal einer bestimmten Klasse in ein Signal einer anderen Klasse zu wandeln. Besondere Bedeutung hat hierbei die Wandlung vom kontinuierlichen zum diskreten Fall, die so genannte Digitalisierung. Dieser Vorgang wandelt zeit- und wertkontinuierliche Signale in digitale Signale um. Das Verfahren ermöglicht beispielsweise die Speicherung von Sprache und Musik auf einer CD.

1.2 Syntax und Semantik

Signale zur Darstellung von Mitteilungen, Inschriften zur Speicherung und die Verarbeitung von Signalen bzw. Inschriften werden allgemein als (technische) Gegenstände bezeichnet. Diese Gegenstände sind für die Informatik zwar relevant, allerdings reichen gewisse gemeinsame Eigenschaften der Gegenstände aus, um die Bedeutung zu erfassen. Die Einschränkung auf bestimmte Gegenstände heißt Abstraktion.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Nachricht
  - Mitteilung, bei der vom verwendeten Medium (Signale und Inschriften) und den Einzelheiten der Signale und Signalparameter abstrahiert wird
  - die Informatik konzentriert sich auf gewisse Eigenschaften von Signalen, Inschriften und deren Verarbeitung

- Information
  - ist die einer Nachricht zugeordnete Bedeutung
  - für die Zuordnung von Bedeutung wird ein Bezugssystem benötigt
  - wird gewonnen durch die Interpretation von Nachrichten auf der Grundlage eines Bezugssystems

Information 4: Nachricht und Information

Der in Information 4 behandelte Übergang von einer Nachricht zu einer Information ist eine wesentliche Frage, die sich die Informatik stellt. Hierdurch wird den die Nachricht bildenden Symbolen eine Bedeutung zugewiesen – es wird der wichtige Übergang von der Syntax zur Semantik hergestellt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Nachricht -α-> Information
  - α ist die Interpretationsvorschrift
  - Interpretationsvorschriften sind selbst Informationen, die durch Nachrichten repräsentiert werden

- Der Satz "Person P versteht die Sprache S" ist mittels der Interpretationsvorschrift zu präzisieren (Hinweis: S ist als eine Menge von Nachrichten aufzufassen)

  _____

- Das Paar (Nachricht, zugeordnete Information) wird als Datum bezeichnet

- Für dieselbe Nachricht können unterschiedliche Interpretationsvorschriften bestehen
  - Beispiel 1: Position einer Person _____
  - Beispiel 2: x^2 _____

Interaktion 5: Interpretationsvorschrift α

Eine Interpretationsvorschrift α erhält als Eingabe die Nachricht und liefert als Ausgabe die Information. Interpretationsvorschriften sind ihrerseits ebenfalls Informationen, die durch Nachrichten repräsentiert werden. In Interaktion 5 soll der Zusammenhang zwischen der Interpretationsvorschrift und dem Satz Person P versteht Sprache S beschrieben werden.

Das durch die Interpretationsvorschrift erzeugte Paar wird auch als Datum bezeichnet. Unter Datenverarbeitung wird die Verarbeitung von Daten unter Einhaltung des Bezugs, der durch die Interpretationsvorschrift vorgegeben wird.

Anhand der in Interaktion 5 angegebenen Beispiele ist aufzuzeigen, dass die Interpretation einer Nachricht, also die Zuordnung einer Bedeutung, nicht immer eindeutig ist.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Wissen ist die Information bzw. Kenntnis, wie Nachrichten zu interpretieren sind

- Zwei Arten von Wissen
  1. Faktenwissen
     - unmittelbare Kenntnis der durch ein Datum gegebenen Information
  2. Synthetisches oder prozedurales Wissen
     - Kenntnis von Interpretationsvorschriften zur Erzeugung solcher Information

- Weitere Eigenschaften von Wissen
  - statisch - dynamisch
  - scharf - unscharf - ungenau
  - detailliert - grob

Information 6: Wissen

Eine besondere Art von Information ist das Wissen. Durch Information 6 wird der Begriff des Wissens eingeführt. Wissen kann sich unmittelbar aus den Daten ergeben (Faktenwissen) oder es wird eine entsprechende Kenntnis zur Erzeugung von Wissen (prozedurales Wissen) benötigt. Wissen selbst ist in Form von geeigneten Daten darzustellen. Die Wissensrepräsentation erfordert also geeignete Datenstrukturen zur Darstellung von Wissen.

Aufgrund der folgenden Kriterien lässt sich Wissen klassifizieren:

- **Kriterium Zeit**
  Aufgrund des Zeitkriteriums lässt sich statisches und dynamisches Wissen unterscheiden. Dynamisches Wissen beschreibt im Gegensatz zu statischem Wissen zeitlich veränderliche Gegenstände und Sachverhalte zu den jeweiligen Zeitpunkten.

- **Kriterien Genauigkeit und Granularität**
  Zwischen diesen Kriterien besteht folgender Zusammenhang: Häufig kann ein Sachverhalt, wie z.B. die Länge eines Holzstücks, aufgrund der Messgenauigkeit nicht exakt angegeben werden. Man muss sich auf eine gewisse Granularität der Angabe einschränken – in diesem Fall die Maßgenauigkeit der Messeinrichtung. Man spricht dann von einem unscharfen Wissen.

www.cm-tm.uka.de/info1
Prof. Abeck & Info1-Team

- Syntax oder syntaktische Struktur von Daten
  - Beziehungen, die sich unmittelbar aus der Anordnung der Zeichen ergeben
- Semantik
  - Weitergehende Beziehungen, die sich aus der Interpretationsvorschrift ergeben
- Pragmatik oder pragmatische Bedeutung
  - Wissen, das sich ergibt, indem man Daten zu Gegenständen oder Sachverhalten außerhalb der vorgegebenen Datenmenge in Beziehung setzt
- Semiotik ist die Lehre von den Zeichen und setzt sich aus den drei obigen Teilen zusammen

Information 7: Semiotik

Ziel der Interpretation ist es, Wissen über Gegenstände innerhalb und außerhalb der Informatik zu gewinnen und zu formulieren. Eine Interpretationsvorschrift gilt als konsistent, wenn sich keine Einsichten aus den Daten herauslesen lassen, die dem darzustellenden Wissen widersprechen.
Falls sich die Daten nicht zur Darstellung eines bestimmten Wissens eignen, handelt es sich um eine so genannte inhärente Interpretationsvorschrift. Man sagt dann auch, die Daten "sind kein Modell" für das Wissen. Die Frage, ob Daten ein Modell für irgendwelches Wissen bilden, ist ausschließlich auf syntaktischer Ebene zu beantworten, da nur die Syntax der Daten beobachtbar und messbar ist.

Die Syntax ist der erste Teil der als Semiotik bezeichneten Lehre der Zeichen, deren weiteren zwei Teile – die Semantik und die Pragmatik – in Information 7 beschrieben sind.

1.3 Codierung und Shannonsche Informationstheorie

Die Frage, wie sich Informationen in Form von Zeichen als Nachrichten darstellen lassen, wird durch die in Interaktion 8 behandelte Codierung beantwortet.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Codierung ist die Darstellung einer Information in Form von Zeichen
  - der für die Codierung verwendete Zeichenvorrat heißt Code
- Beispiel: Codierung der Information zwei
  1. _____ Code: _____
  2. _____ Code: _____
  3. _____ Code: _____
- Redundante Codierung ermöglicht das Erkennen von fehlender oder verfälschter Information aufgrund der bestehenden codierten Information
- Binärcode
  - Zeichenvorrat, der aus zwei Werten, den Binärzeichen besteht
  - Beispiel: B = {O, L}
  - beliebige Zahlwerte hiermit codierbar

Interaktion 8: Codierung von Informationen

Eine Codierung basiert immer auf einem Zeichenvorrat, dem Code. In dem in Interaktion 8 nachgefragten Beispiel wird der Code alternativ
- aus Folgen lateinischer Buchstaben oder
- aus einer Dezimalziffer oder
- aus Strichen gebildet.

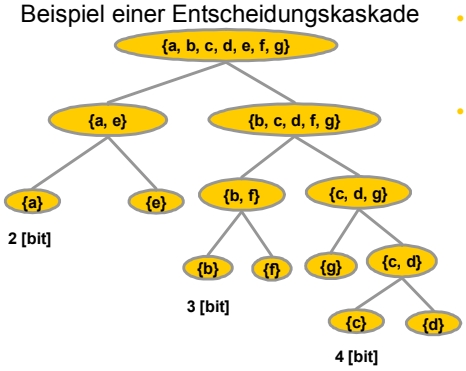
Ein Beispiel für einen redundanten Code ist die Verkehrsampel. Bei Ausfall eines der Lichter lässt sich das Signal aufgrund der zwei noch funktionierenden Lichter rekonstruieren. In der Telekommunikation erreicht man einen redundanten Code dadurch, dass zu der Codierung noch geeignete Prüfinformation hinzugefügt wird.

Eine Antwort, wie eine "optimale" Codierung von Zeichen eines vorgegebenen Zeichenvorrats aussieht, liefert die Shannonschen Informationstheorie [BG73].

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Kern der Theorie: Bei Eintreffen eines von einer Shannonschen Quelle (Nachrichtenquelle) gesendeten Zeichen ist zu entscheiden, welches Zeichen eines im voraus bekannten Zeichenvorrats vorliegt
  - das Ziel ist, für häufig auftretende Zeichen die Anzahl der dafür aufzuwendenden Entscheidungen möglichst klein zu halten
  - das Ziel ist durch geeignete Zerlegung in "gleichwahrscheinliche" Teilmengen zu erreichen

Beispiel einer Entscheidungskaskade



- p_i sei die Wahrscheinlichkeit, dass das i-te Zeichen auftritt (a ist 1. Zeichen, b ist 2. Zeichen,..., g ist 7. Zeichen)
- Wie groß sind die Wahrscheinlichkeiten für die im Beispiel auftretenden Zeichen unter der Annahme, dass eine exakte Gleichheit der Teilmengen gegeben ist?

p_1 = _____ p_2 = _____
p_3 = _____ p_4 = _____
p_5 = _____ p_6 = _____
p_7 = _____

Interaktion 9: Shannonsche Informationstheorie

Von C. Shannon wurde im Jahr 1948 eine Theorie entwickelt, durch die der Entscheidungsgehalt eines Zeichens mathematisch gefasst werden kann.

Der Theorie, dessen Kern in Interaktion 9 beschrieben ist, geht dabei von einer elementaren Entscheidung zwischen zwei Zeichen aus. Jede solche Alternativentscheidung wird in der Einheit [bit] gemessen. Der Zusammenhang, der zwischen dieser Maßeinheit [bit] (mit kleinem b) und dem gerade eingeführten Bit (mit großem B) wird weiter unten geklärt.

Anhand der für ein einfaches Beispiel aufgestellten Entscheidungskaskade wird das Vorgehen, das der Shannonschen Informationstheorie zugrunde liegt, deutlich: Ist ein Zeichen zu bestimmen, so steigt man die Entscheidungskaskade schrittweise durch die Alternativentscheidung ab, ob das Zeichen in der einen oder anderen Teilmenge liegt. Der Entscheidungsgehalt des Zeichens ist durch die Anzahl der erforderlichen Alternativentscheidungen bestimmt.

Das Ziel der Zerlegung, die der Entscheidungskaskade zugrunde liegt, besteht darin, dass die Summen der Wahrscheinlichkeiten der in den Teilmengen enthaltenen Zeichen für die zwei Teilmengen möglichst genau gleich sind. Im Beispiel wird von der (in realen Zeichenvorräten nicht gültigen) Annahme ausgegangen, dass eine exakte Gleichheit erzielt werden kann. Aus der Entscheidungskaskade lassen sich die Wahrscheinlichkeiten der Zeichen unter dieser Annahme einfach ermitteln (siehe Interaktion 9).

- Zusammenhang zwischen Wahrscheinlichkeit des Auftretens des i-ten Zeichens p_i und der für dieses Zeichen benötigten Alternativentscheidungen-Anzahl k_i

  $p_i = (½)^{k_i}$

- Daraus ergibt sich der Entscheidungsgehalt eines Zeichens

  $k_i =$ _____ [bit]

- Entscheidungsgehalt H eines Zeichenvorrats
  - $H = \Sigma\ p_i * ld\ (1/p_i)$ [bit]
    - ist der mittlere Entscheidungsgehalt pro beliebig herausgegriffenem Zeichen

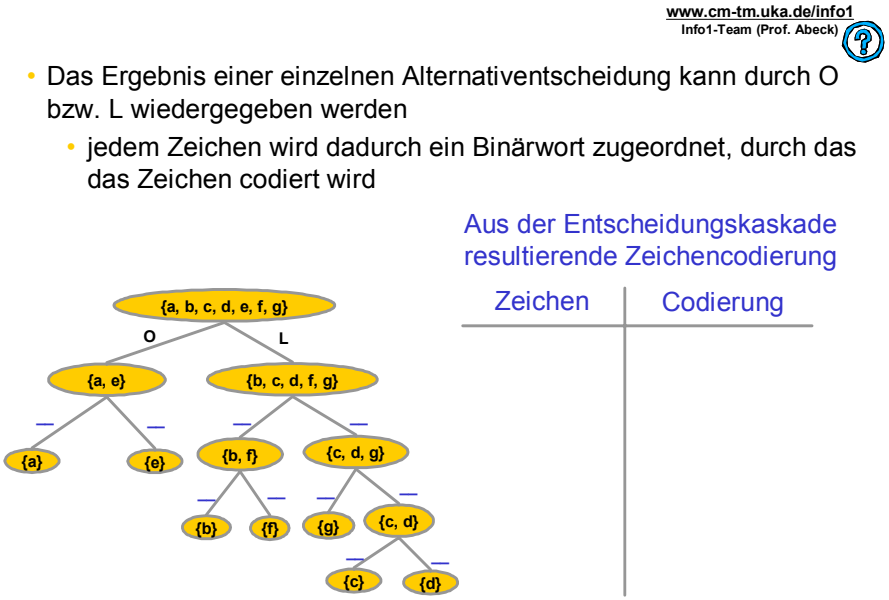
- Entscheidungsgehalt eines Zeichenvorrats mit n Zeichen
  - für n = 2 Zeichen gilt H = _____
  - für n > 2 Zeichen gilt $H \leq$ _____

Interaktion 10: Entscheidungsgehalt

Unter der Annahme der exakten Gleichheit besteht offensichtlich der in Interaktion 10 dargestellte Zusammenhang zwischen der Wahrscheinlichkeit des Auftretens eines Zeichens und der Anzahl an Alternativentscheidungen, die zu dessen Auswahl erforderlich sind. Diese Gleichung lässt sich einfach so umformen, dass eine Formel für die Anzahl der Alternativentscheidungen gewonnen werden kann.

Diese Anzahl wird der Entscheidungsgehalt eines Zeichens genannt. Die Messeinheit ist die eingeführte Einheit [bit]. Der Entscheidungsgehalt eines Zeichens bildet die Grundlage zur Ermittlung des Entscheidungsgehalts des Zeichenvorrats, in dem dieses Zeichen enthalten ist, wie den Ausführungen in Interaktion 10 entnommen werden kann.

Die Größe H und deren Bestimmung bilden die Basis der Shannonschen Informationstheorie. H wird als mittlerer Entscheidungsgehalt pro Zeichen oder als Information pro Zeichen bezeichnet.



- Das Ergebnis einer einzelnen Alternativentscheidung kann durch O bzw. L wiedergegeben werden
  - jedem Zeichen wird dadurch ein Binärwort zugeordnet, durch das das Zeichen codiert wird

Interaktion 11: Codierung von Zeichen

Interaktion 11 zeigt den Entscheidungsbaum, in dem jeder linke Ast mit O und jeder rechte Ast mit L zu markieren ist und die resultierende Codierung der Zeichen anzugeben ist.

Im Zusammenhang mit der Codierung erhält die Größe H eine weitere Bedeutung, nämlich die mittlere Wortlänge des Codes, der in der Einheit Bit gemessen wird. Hiermit ist dann auch der Zusammenhang zwischen [bit] und Bit hergestellt.
Durch die Codierung werden die Zeichen des Zeichenvorrats in eine Reihenfolge gebracht, was zum Begriff des Alphabets führt.

- Ein Zeichenvorrat, in dem eine Reihenfolge (lineare Ordnung) für die Zeichen definiert ist, heißt ein Alphabet

- Beispiele für Alphabete
  - Alphabet der Dezimalziffern {0, 1, 2, 3, 4, 5, 6, 7, 8 ,9}
  - Alphabet der lateinischen Buchstaben

- Die wichtigsten für die Zeichenverarbeitung benötigten Alphabete sind Bestandteil des ASCII-Zeichenvorrats
  - 7-Bit-Codierung
  - wird erweitert durch den Unicode (16-Bit-Codierung)
  - auf den ASCII-Code und Unicode wird im Zusammenhang mit der Behandlung der Sprache Java ausführlich eingegangen

Information 12: Alphabet und Codes

In einem Alphabet ist eine Reihenfolge auf den Zeichen definiert. Diese Zeichenreihenfolge ermöglicht eine lexikographische Anordnung von Zeichenreihen. Hierzu sind allerdings noch gewisse Zusatzinformationen erforderlich. So muss beispielsweise beim lateinischen Alphabet der Umgang mit den Umlauten geklärt werden. Die Anordnung nach DIN 5007 sagt aus, dass die Umlaute ä, ö, ü wie a, o, u zu behandeln sind. Diese Regelung gilt beispielsweise nicht bei

der Ordnung von Namen im Telefonbuch, in dem Umlaute wie Ligaturen ae, oe, ue behandelt werden.

Ein bekannter, in der Informatik weit verbreiteter Code ist der ASCII-Code. ASCII steht für *American Standard Code for Information Interchange* und ist eine Ausprägung des ISO-7-Bit-Codes.

Auf den ASCII-Code und dessen Erweiterung zum so genannten Unicode wird in einer späteren Kurseinheit PROGRAMMIERGRUNDLAGEN im Zusammenhang mit der Programmiersprache Java und dem darin bereitgestellten Zeichentyp (char) näher eingegangen.

2 ALGORITHMUS

Ein weiterer zentraler Begriff der Informatik neben der Information ist der Algorithmus. Es besteht ein enger Zusammenhang zwischen diesen beiden Grundbegriffen, wie im Folgenden näher ausgeführt wird [Go97].

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ziel der Informationsverarbeitung ist die Verknüpfung von Informationen unterschiedlicher Herkunft zur Gewinnung neuer Informationen
- Zwei verschiedene Formen der Verknüpfung lassen sich unterscheiden
  1. transformierende (selektierende) Informationsverarbeitung
     - die verknüpften Informationen gehen in das Ergebnis auf und lassen sich nicht oder nur noch teilweise zurückgewinnen
  2. strukturierende (relationale) Informationsverarbeitung
     - zwischen den Eingangsdaten werden Beziehungen hergestellt, die Bestandteil des Ergebnisses sind
- Aus technischer Sicht werden der Informationsverarbeitung Eingangsdaten zugeführt
  - Ergebnis der Transformation bzw. Strukturierung liefern die Ausgangsdaten
  - Informationsverarbeitung entspricht einer Funktionsberechnung

Information 13: ALGORITHMUS - Informationsverarbeitung

In Information 13 werden zwei Formen der Informationsverarbeitung unterschieden. Beispiele für eine transformierende Informationsverarbeitung ist die Durchführung einer Addition oder die Bestimmung der n-ten Primzahl. Eine strukturierende Informationsverarbeitung liegt beispielsweise vor, wenn Wörter (bestehende Information) zu einem Satz (neue Information) zusammengefügt werden.

Die Verarbeitung von Informationen setzt entsprechende Eingangsdaten bzw. die sie repräsentierenden Signale voraus. Die durch Transformation oder Strukturieren entstehende Information schlägt sich in entsprechenden Ausgangsdaten nieder.

Die Informationsverarbeitung kann als eine Funktionsberechnung aufgefasst werden, die die Eingangsdaten in Ausgangsdaten überführt. Eine Funktionsberechnung kann auf drei verschiedenen Arten erfolgen, was in Information 14 zum Begriff des Algorithmus führt.

- statisch (deklarativ)
  - Benutzung von Gleichungen
  - kann zu einem konstruktiven Verfahren weiterentwickelt werden

- tabellarisch
  - Aufstellen einer Tabelle, in der für alle Werte a des Definitionsbereichs A der Wert f(a) im Wertebereich B angegeben ist
  - nicht anwendbar für unendliche bzw. zu große Definitionsbereiche

- algorithmisch (operativ, prozedural, synthetisch)
  - Beschreibung zur Berechnung des Ergebnisses für beliebige Argumente a des Definitionsbereichs A
  - die in der Beschreibung verwendeten elementaren Operationen müssen hinreichend präzise spezifiziert sein

Information 14: Arten der Berechnung einer Funktion f:A→B

statisch oder deklarativ: Die Zusammenhänge zwischen dem Definitionsbereich A und dem Wertebereich B werden statisch durch Gleichungen angegeben. Im ersten Moment erscheint es, dass hierdurch kein konstruktives Verfahren beschrieben wird, sondern nur die Möglichkeit gegeben wird, die Richtigkeit eines Resultats zu überprüfen. Durch die in der Kurseinheit RECHENSTRUKTUREN näher beschriebenen Termersetzungsverfahren kann aus der deklarativen Beschreibung eine konstruktive Berechnungsmethode hergeleitet werden.

tabellarisch: hierbei handelt es sich um die direkteste Form der Angabe einer Funktion, die aber nur in seltenen und besonders einfachen Fällen zur Anwendung kommen kann. Insbesondere muss erfüllt sein, dass der Definitionsbereich A nicht allzu groß ist. Der Berechnungsvorgang besteht darin, in der Tabelle das gewünschte Element aus dem Definitionsbereich zu suchen und den entsprechenden Tabelleneintrag des Wertebereichs auszugeben.

algorithmisch: Folge von endlich vielen elementaren Operationen, durch die das Vorgehen zur Ermittlung des gewünschten Funktionsergebnisses hinreichend präzise beschrieben ist. Algorithmen bilden die Grundlage jedes Informatiksystems.

2.1 Euklidischer Algorithmus

Ein berühmtes Beispiel eines Algorithmus, also eines mechanisch ausführbaren Rechenverfahrens, ist der so genannte Euklidische Algorithmus. Seinen Namen hat dieser Algorithmus von Euklid, einem griechischen Mathematiker (um 300 v. Chr.). Der Euklidsche Algorithmus ist der älteste und damit erste Algorithmus überhaupt. Ein Vergleich mit der Jahreszahl von Al Khwarizmi (um 800 n. Chr.), nach dem der Begriff des Algorithmus benannt ist, macht deutlich, dass der Begriff Algorithmus zu dieser Zeit noch gar nicht bekannt war.

- Aufgabe: Bestimmung des größten gemeinsamen Teilers (ggT) aus zwei gegebenen natürlichen Zahlen p und q

- Die zwei Schritte des Euklidischen Algorithmus
  1. Dividiere p durch q. Dabei erhält man einen Rest r, der zwischen 0 und q-1 liegt.
  2. Wenn r=0 ist, dann ist q der gesuchte größte gemeinsame Teiler.
     Wenn r<>0, dann benenne man das bisherige q in p sowie das bisherige r in q um und wiederhole Schritt 1 und Schritt 2 so lange, bis r=0 geworden ist.

- Beispiel: Der ggT(24, 18) ist gemäß dem angegeben Verfahren zu berechnen

Interaktion 15: Euklidischer Algorithmus

Wie in Interaktion 15 beschrieben wird, liefert der Euklidische Algorithmus zu zwei natürlichen Zahlen den größten gemeinsamen Teiler. Der größte gemeinsame Teiler ggT(p, q) ist die größte Zahl, die p und q ohne Rest teilt.

Der Euklidische Algorithmus berechnet den ggT in zwei, ggf. mehrfach ausgeführten Schritten. Zunächst wird p durch q dividiert, wobei es sich hier um eine ganzzahlige Division handelt. Das bedeutet, dass die Division keine Nachkomma-Stellen berechnet, sondern einen Rest erzeugt, falls die Division nicht aufgeht. Schritt 2 beginnt mit der Endebedingung des Algorithmus: Man ist fertig, wenn die ganzzahlige Division von Schritt 1 aufgegangen ist, d.h. keinen Rest ergeben hat. Andernfalls wiederholt man das Verfahren (allerdings mit den im letzten Durchlauf ermittelten Werten).
Das Beispiel macht deutlich, dass der Ablauf des Euklidischen Algorithmus durch die in den Berechnungsgrößen (hier r, p, q) enthaltenen Werte abhängt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Die Ablaufsteuerung legt die Reihenfolge und Ausführungsbedingungen von Tätigkeiten (elementaren Operationen) in einem Algorithmus fest
- Ablaufsteuerungselemente
  - sequentielle Ausführung
  - parallele bzw. kollaterale Ausführung
  - bedingte Ausführung
  - Ausführung in einer Schleife
  - Ausführung eines Unterprogramms
- Hiermit können beliebige Abläufe in algorithmischer Form beschrieben werden
- Welche Ablaufsteuerungselemente wurden im Euklidschen Algorithmus genutzt?

_____

_____

_____

Interaktion 16: Ablaufsteuerung

Es werden die in Interaktion 16 genannten Elemente der Ablaufsteuerung unterschieden. Zu jedem der aufgeführten Ablaufsteuerungselemente werden in den nachfolgenden Kurseinheiten entsprechende Anweisungen der Programmiersprache Java eingeführt, die zur Formulierung von Algorithmen dienen, die auf einem Rechner ausgeführt werden können. Man spricht dann von Programmen.

2.2 Effizienz von Algorithmen

Bei einem Algorithmus stellt sich grundsätzlich die in Information 17 angesprochene Frage nach seiner Effizienz [BB93].

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Mit der Effizienz ist unmittelbar die Frage verbunden, welcher von mehreren vorgegebenen Algorithmen zur Lösung eines Problems vorzuziehen ist
- Zwei unterschiedlich Vorgehensweisen zur Beantwortung der Frage

  (1) Empirisches (a posteriori) Vorgehen

  Programmieren der Algorithmen und Erprobung mit verschiedenen Fällen (Eingaben) auf einem Rechner

  (2) Theoretisches (a priori) Vorgehen

  Bestimmung des Ressourcenverbrauchs eines Algorithmus (Ausführungszeit, Speicherplatz, ...) als eine Funktion der Größe der betrachteten Fälle

Information 17: Effizienz von Algorithmen

Der Vorteil einer theoretischen Näherung des Ressourcenverbrauchs gegenüber dem empirischen Vorgehen besteht darin, dass die Effizienz eines Algorithmus unabhängig
- vom eingesetzten Rechner
- von der verwendeten Programmiersprache oder
- vom Geschick des Programmierers

bestimmt werden kann.

Die Größe eines Falles, die bei der Bestimmung der theoretischen Effizienz zu ermitteln ist, beschreibt den Umfang der Darstellung dieses Falls auf einem Rechner. Formal handelt es sich dabei um eine Anzahl von Bits. In einer konkreten Analyse interessiert die Anzahl der Komponenten eines Falls – auch wenn die Komponenten zur Darstellung üblicherweise mehr als ein Bit benötigen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Die Ordnung von f(n), also O(f(n)), ist die Menge aller Funktionen t(n), die nach oben durch ein positives reelles Vielfaches von f(n) begrenzt werden, vorausgesetzt n ist genügend groß (größer als eine Schwelle n_0)

- Asymptotische Schreibweise der Ordnung
  - $O(f(n)) = \{t: \mathbb{N} \rightarrow \mathbb{R}^* \mid (\exists c \in \mathbb{R}^+) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) [t(n) \leq c*f(n)]\}$
    - $\mathbb{N}$ natürliche Zahlen, $\mathbb{R}^*$ nicht-negative reelle Zahlen
    - $f: \mathbb{N} \rightarrow \mathbb{R}^*$ beliebige Funktion

Information 18: Ordnung und asymptotische Schreibweise

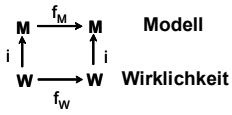
Durch die in Information 18 eingeführte asymptotische Schreibweise wird erreicht, die multiplikative Konstante angemessen in die theoretischen Analysen von Algorithmen einzubeziehen.

3 MODELL

Von besonderer Bedeutung in der Informatik ist der Vorgang des Modellierens, d.h. die Abbildung und Darstellung der Wirklichkeit in einem Modell [Go97].

- Wirklichkeit
  - Dinge, Personen und deren Beziehungen
  - Abläufe in der Zeit
- Modell
  - Begriffe von Dingen und Personen und Beziehungen zwischen diesen Begriffen
  - Abläufe in gedachter Zeit
- Ein Modell ist wahr, wenn die Begriffe die Wirklichkeit richtig wiedergeben

i: Gegenstände der Wirklichkeit ? Begriffe der Modellwelt
f_M: beliebige Beziehung zwischen Begriffen der Modellwelt
f_W: reale Beziehung



Information 19: MODELL – Beziehung zwischen Wirklichkeit und Modell

Das Ziel muss darin bestehen, richtige oder wahre Modelle aufzubauen. Das Modell sollte also die Sachverhalte der Wirklichkeit richtig wiedergeben. Diese Forderung lässt sich durch die Einführung von Abbildungen formal ausdrücken.

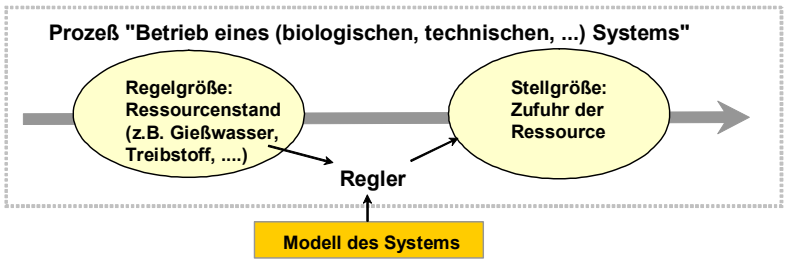
Die Aussage "das Modell ist wahr" bedeutet, dass Beziehungen, die zwischen Gegenständen (bzw. deren Zuständen) innerhalb des Modells bestehen, auch für die entsprechenden Gegenstände innerhalb der Wirklichkeit bestehen müssen. Man sagt dann auch, dass das Beziehungsdiagramm zwischen Modell und Wirklichkeit kommutiert.

3.1 Ziele von Modellen

Ein Beispiel eines Modells ist ein Plan, der von einem Statiker zu einer in Zukunft zu erstellenden Brücke angefertigt wird. Das Modell soll verhindern, dass die gemäß dem Plan gebaute Brücke einstürzt. Mit dem Modell wird somit eine Vorhersage auf zukünftiges Verhalten ermöglicht.

- Gewinn von Einsichten in Vergangenes und Bestehendes sowie Ermöglichung von Vorhersagen über zukünftiges Verhalten

- Die Informationsverarbeitung greift auf der Grundlage der aus den Modellen abgeleiteten Aussagen steuernd oder regelnd in die Wirklichkeit ein



- Welches Ziel wird im Beispiel mit dem Modell verfolgt?

- Wovon hängt die Qualität des Ergebnisses ab?

Interaktion 20: Mit Modellen verfolgte Ziele

Modelle können auch dazu genutzt werden, auf die reale Welt regelnd einzuwirken. Ein Beispiel ist die Regelung einer Ressource, die von einem technischen System für dessen Betrieb benötigt wird, wie die Abbildung in Interaktion 20 zeigt. Konkret könnte das technische System ein Kraftfahrzeug sein und die Ressource der Treibstoff. Ein anderes Beispiel ist das Gießen eines Gartens. In diesem Fall ist das System der Garten (also ein biologisches System), die Ressource ist das Gießwasser und der Regler ist der Gärtner.

Der bislang behandelte Begriff der Regelung unterscheidet sich vom Begriff der Steuerung. Regelung beinhaltet immer eine Rückkopplung zwischen Regler und System. Fehlt ein solcher Rückkopplungsmechanismus, so handelt es sich um eine Steuerung.

- Abstraktion eines Modells bedeutet Weglassen von Details
  - Ziel: Vereinfachung der Informationsverarbeitung
  - Frage: entspricht das abstrahierte Modell noch der Wirklichkeit?

- Validierung eines Modells bedeutet die Überprüfung des Wahrheitsgehalts dieses Modells
  - kann nur durch Experimente und nicht durch logische Schlüsse in der Modellwelt erfolgen

- Spezifikation ist die Beschreibung einer Wirklichkeit
  - ist dann gegeben, wenn die Wirklichkeit einer Gedankenwelt entspringt

- Verifikation ist die Validierung eines Modells gegen eine Spezifikation
  - basiert ausschließlich auf logischen Schlüssen
  - in der Praxis stellt sich das Problem der (zu) hohen Komplexität der Prüfaufgabe

Information 21: Arbeiten mit Modellen

Aufgrund der Komplexität der Wirklichkeit ist man in der Informationsverarbeitung häufig gezwungen, das Modell zu abstrahieren bzw. zu vereinfachen (siehe Information 21). Hierdurch kann es zu einem Realitätsverlust kommen, der Ursache für Fehler und somit ungünstige steuernde Eingriffe sein kann.

Daher ist es wichtig, das Modell auf seinen Wahrheitsgehalt zu überprüfen. Eine solche Überprüfung wird Validierung genannt. Da die Überprüfung in der Wirklichkeit stattfindet, sind hierfür Experimente durchzuführen.

Auf der Grundlage von logischen Schlüssen kann nur dann der Wahrheitsgehalt eines Modells überprüft werden, wenn die Wirklichkeit unserer Gedankenwelt entspringt und als Spezifikation vorliegt. Diese Form der Überprüfung wird als Verifikation bezeichnet.

Modelle werden immer vom Menschen erdacht, er trägt die Verantwortung für sämtliche Interpretationsschritte. Informatiker sollten daher immer verantwortungsvoll mit Modellen umgehen (siehe Information 22).

- Konstruierte und eingesetzte Modelle können zur Veränderung der Wirklichkeit führen
  - Beispiel Telekommunikationssysteme: Das Modell legt die (technische) Kommunikation zwischen den Teilnehmern fest
- Der Konstrukteur sollte sich die Konsequenzen, die durch den regelnden und steuernden Eingriff in die Wirklichkeit entstehen, bewußt machen
- Kein Modell oder Verfahren der Informatik ist von vornherein gut oder schlecht
  - es kommt auf deren Einsatzzweck und Einsatzform an
  - Beispiel: Datenbanken und Datenschutzaspekte

Information 22: Verantwortungsvoller Umgang mit Modellen

Der Informatiker hat sich speziell damit auseinander zu setzen, welche Auswirkungen der Einsatz von Rechnern und Telekommunikation auf die Menschheit hat. Hierin liegen Chancen (z.B. Einsparung von Energie durch Tele-Arbeit), aber auch Gefahren (z.B. Vereinsamung, gläserner Mensch).

3.2 Beispiel eines Modells

Anhand des in der Kurseinheit EINFÜHRUNG IN DAS FACHGEBIET eingeführten Beispielszenarios des Internet-basierten Wissenstransfers wird nachfolgend aufgezeigt, wie ein Teil der Wirklichkeit konkret in ein Informatik-gerechtes Modell überführt werden kann. Dabei wird hier eine objektorientierten Modellierung verfolgt, die eine große Bedeutung im Zusammenhang mit der Entwicklung von Informatiksystemen hat. Als Modellierungssprache kommt die allgemein akzeptierte und standardisierte *Unified Modeling Language* (UML, [Oe01]) zum Einsatz.

Auf die Konzepte der Objektorientierung wird detailliert im Rahmen der Kurseinheit OBJEKTORIENTIERTE PROGRAMMIERUNG eingegangen. In diesem Abschnitt wird das Ziel verfolgt, die bislang eher abstrakt beschriebene Aufgabe der Modellierung zu präzisieren.

- Die Ausführungen zum Modell werden im Folgenden an dem Beispiel-Szenario des Internet-basierten Wissenstransfers konkretisiert

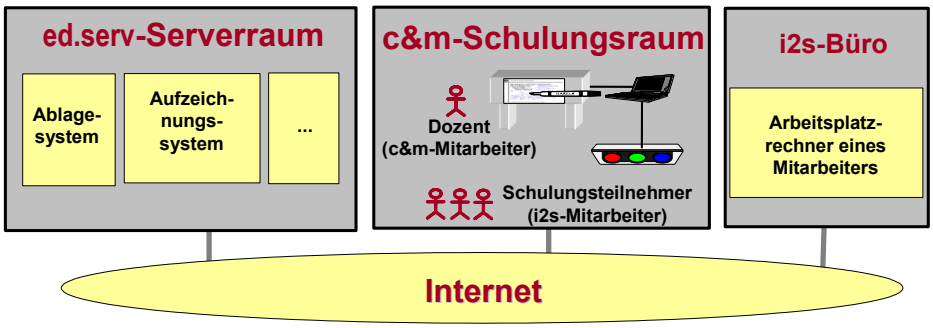
- Es wird ein objektorientierter Modellierungsansatz verfolgt
  - als Modellierungssprache wird die UML (Unified Modeling Language) genutzt

- Ein Objekt beinhaltet die Information, die bzgl. des Gegenstandes der Wirklichkeit innerhalb des Modells benötigt wird

- Alternative Formen der Verarbeitung
  1. innerhalb eines Objekts
  2. durch Kommunikation von Nachrichten zwischen zwei Objekten

Information 23: Objektorientierter Modellierungsansatz

Das Objekt ist ein Informationsträger innerhalb des Modells, wie in Information 23 ausgeführt ist. Die Verarbeitung der Information erfolgt entweder innerhalb des Objekts oder zwischen zwei Objekten, indem die Objekte untereinander Nachrichten austauschen [BH03].



- Das Schulungsunternehmen c&m nutzt die vom IT-Dienstleister ed.serv angebotenen Dienste des Internet-basierten Wissenstransfers, um i2s einen Schulungsdienst anzubieten

- Obige Darstellung beschreibt die Wirklichkeit auf einem informellen Niveau
  - schlägt die Brücke zum Anwender (der "Kunde" des Informatikers)
  - Einstieg in ein im Folgenden weiter zu detaillierendes Modell

Information 24: Beispielszenario

Information 24 zeigt das bereits bekannte Beispielszenario. Die Abbildung kann als eine wirklichkeitsnahe Skizze aufgefasst werden. Die Darstellung ist bewusst frei von fachspezifischen Modellelementen gehalten, damit die Problemstellung und deren Lösung mittels eines Informatiksystems dem künftigen Anwender dieses Informatiksystems vermittelt werden kann.

Für eine weitergehende Modell-Verfeinerung, die ab einer gewissen Detaillierungsstufe nur noch von einem Informatiker – dem Entwickler der Systemlösung – verstanden wird, bietet sich die graphische Modellierungssprache UML an.

Das Identifizieren der im Beispielszenario auftretenden Objekte ist ein wichtiger Schritt der Modellverfeinerung, der nachfolgend an einem Ausschnitt aus dem Beispielszenario durchgeführt wird.

- Kandidaten für Objekte
  - Personen, Gebäude, Räume, Geräte, Dokumente, ...

- Es werden nur ganz bestimmte Informationen zu den realen Gegenständen in dem Objekt gehalten
  - entscheidend ist das mit dem Modell verfolgte Ziel

Interaktion 25: Auftretende Objekte im Beispielszenario

Im vorgegebenen Szenario lassen sich problemlos Objekte identifizieren, durch die reale innerhalb einer c&m-Schulung auftretende "Gegenstände" modelliert werden. Es sollen in Interaktion 25 einige solcher Objekte exemplarisch durch Angabe der Art (Personen, Gebäude, Geräte, Dokumente, ...) und beliebig gewählter Namen genannt werden.

Durch das Hinzufügen von Daten (Attributen) und Funktionen (Operationen bzw. Methoden) werden die Objekte zu Informationsträgern bzw. informationsverarbeitenden Einheiten. Die integrierte Behandlung von Information und deren Verarbeitung in einem Objekt ist eine zentrale Charakteristik des objektorientierten Ansatzes.

- Ein Objekt ist durch seinen Zustand und seine Funktionalität bestimmt
  - zur Beschreibung des Zustands dienen Attribute und darin gehaltene Werte
  - die Funktionalität wird durch Operationen, die auch als Methoden bezeichnet werden, beschrieben

- Beispiel: Vorliegendes Kursdokument
  - Attribute
    - Titel: "GRUNDBEGRIFFE DER INFORMATIK"
    - Autor: "Abeck"
    - Seitenanzahl: 34
  - Methoden
    - Wie lautet der Titel? Welcher Autor? Wie viele Seiten?
    - Ändere Seitenanzahl

Information 26: Attribute und Operationen

Die Beschreibung eines Objekts betrifft die zwei in Information 26 genannten Aspekte: Durch die Einführung von Attributen wird das Objekt zu einem Informationsträger. Hierdurch lassen sich alle Aspekte des Objekts beschreiben, die aus der Sicht der Informatiklösung über dieses Objekt bekannt sein sollten. Durch die Angabe von Operationen wird das Objekt zu einer aktiven Einheit, die eine Verarbeitung der in den Attributen oder auch außerhalb des Objekts zugreifbaren Information durchführen kann.

Am Beispiel eines Kursdokuments, das als ein Objekt des dem Beispielszenario zugrunde liegenden Modells identifiziert werden kann, lässt sich die Festlegung von Attributen und Operationen aufzeigen. Es sei betont, dass diese Festlegung ausschließlich von den Anforderungen der zu entwickelnden Informatiklösung abhängt.

Typische Operationen sind das Lesen und Schreiben von Attributwerten des Objekts, wie die in Information 26 genannten Operationen des Beispiel-Objekts verdeutlichen.

Die Zusammenfassung bzw. die "Klassifikation" von Objekten mit gleichen Merkmalen (d.h. Attribute und Methoden) führt zu dem wichtigen Begriff der Klasse.

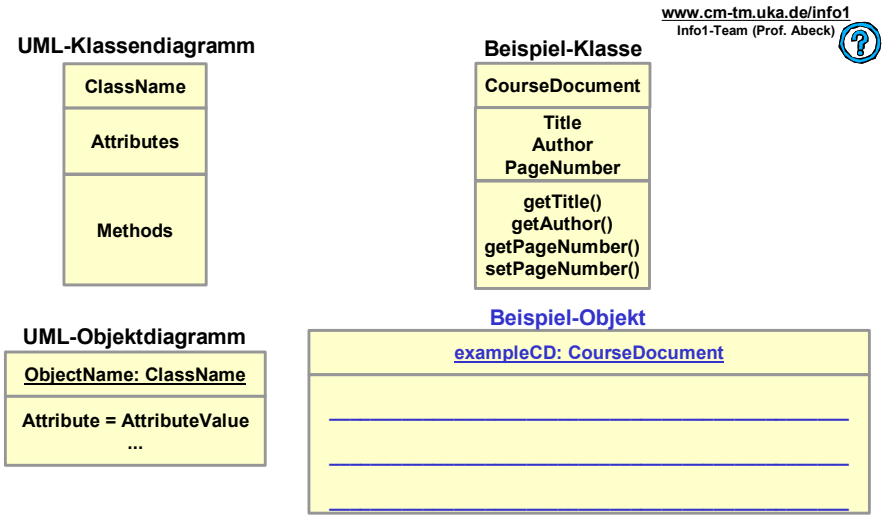
www.cm-tm.uka.de/info1
Prof. Abeck & Info1-Team

- Eine Klasse beschreibt eine Menge von Objekten mit gleichen Merkmalen
- Merkmale eines Objekts sind die Attribute und Methoden dieser Klasse
- Ein Objekt dieser Menge wird auch als Instanz der Klasse bezeichnet
- Eine Klasse beschreibt die Merkmale ihrer Objekte
  - Die Klassenbeschreibung wirkt wie eine Schablone zur Erzeugung (Instanziierung) von ihr zugeordneten Objekten (Instanzen)

Information 27: Klasse

Anhand des Kursdokuments lässt sich der Zusammenhang zwischen einer Klasse und einem Objekt einfach klarmachen: Ein Kursdokument ist eine Klasse, durch die die Merkmale (Attribute und Operationen) der zu dieser Klasse gehörenden Objekte, wie z.B. das Kursdokument mit dem Titel GRUNDBEGRIFFE DER INFORMATIK, beschrieben werden.

Zur Beschreibung von Klassen und Objekten sowie den darin enthaltenen Attributen und Methoden bietet die UML grafische Beschreibungselemente an, durch die eine übersichtliche Darstellung ermöglicht wird.



- Konvention, die für diese Veranstaltung eingeführt wird
  - In entwicklernahen (UML-) Klassen-/Objektdiagrammen sowie in den die Diagramme umsetzenden (Java-) Programmen werden durchgängig englische Bezeichnungen verwendet

Interaktion 28: Klassendiagramm und Objektdiagramm

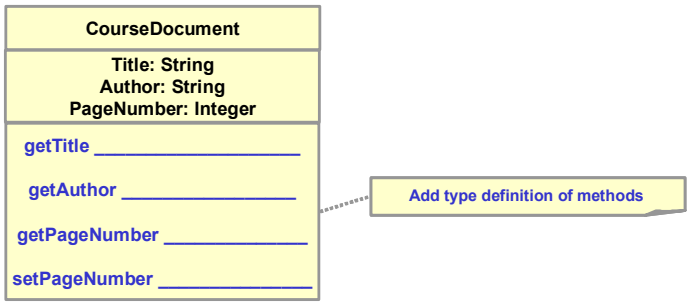
In der UML werden Klassen und Objekte durch ein Rechteck beschrieben, in dem die Elemente der Klassenbeschreibung (Name, Attribute und Operationen) bzw. Objekte (Name und Attributwerte) in einzelnen Abschnitten aufgeführt werden. Interaktion 28 zeigt die als Diagramme bezeichneten grafischen Beschreibungen und liefert für das Klassen- und

Objektdiagramm jeweils ein Beispiel, das das zuvor beschriebene Kursdokument-Objekt aufgreift. Das Objektdiagramm zum Beispiel-Objekt exampleCD ist gemäß den in Information 26 angegebenen Attributwerten zu vervollständigen.

Klassen- und Objektdiagramme lassen sich unmittelbar in einer objektorientierten Programmiersprache, wie z.B. Java umsetzen. Da in den Java-Programmen, die in nachfolgenden Kurseinheiten entwickelt werden, durchgängig die englische Sprache zur Bezeichnung von Klassen, Objekten, Attributen, Methoden, usw. verwendet werden, bietet es sich an, auch in den entwicklernahen UML-Diagrammtypen ausschließlich englischsprachige Begriffe zu verwenden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- In den eingeführten Diagrammen sind die Attribute ausschließlich durch ihre Namen beschrieben
- Das Beispiel zeigt, dass die Werte von einem bestimmten Typ sind
  - Titel und Autor tragen als Werte eine Zeichenkette (String)
  - Seitenanzahl ist eine (positive) ganze Zahl zuzuweisen (Integer)
- Die Klassenbeschreibung kann bei Bedarf um diese Typangaben ergänzt werden
- Beispiel:



Interaktion 29: Ergänzung der Typangabe

Anhand der einfachen Beispiel-Klasse CourseDocument wird bereits offensichtlich, dass Attributen Werte unterschiedlicher Wertemengen zugeordnet sein können. Dieser Sachverhalt wird durch eine Typangabe ausgedrückt, die zu einem Attribut im Klassendiagramm angegeben werden kann. Die Typangabe zu den Methoden ist in Interaktion 29 zu ergänzen.

Beispiele für Typen sind String (Menge der Zeichenketten) oder Integer (Menge der ganzen Zahlen). Auf Typen wird in den nachfolgenden Kurseinheiten detailliert eingegangen.

4 ARCHITEKTUR

In [Oe01] wird die Architektur definiert als die Spezifikation der grundlegenden Struktur eines Systems.

- Es sind verschiedene ineinander greifende Architekturkonzepte zu unterscheiden
  - Rechensystem
    - Architektur eines einzelnen Rechners
    - vorherrschend ist die von-Neumann-Rechnerarchitektur
  - Verteiltes System
    - Architektur eines vernetzten Rechensystems
    - Client-Server-Prinzip
  - Verteilte Anwendung
    - Softwaresystem, das auf den (Client-Server-) Systemen abläuft
    - Mehrschichtenarchitektur

Information 30: ARCHITEKTUR - Überblick

Zunächst wird als System das Rechensystem zugrunde gelegt, dessen Architektur durch den so genannten von-Neumann-Rechner vorgegeben ist.

Rechensysteme werden heute üblicherweise vernetzt (meist über das Internet). Hieraus resultieren Verteilte Systeme, die gemäß einem Architekturprinzip aufgebaut sind, das als Client-Server-Prinzip bezeichnet wird.
Die auf den (Client-Server-) Systemen laufende Anwendungssoftware ist in gewissen Schichten organisiert, wobei jede der Schichten ein bestimmtes Aufgabengebiet in der Anwendung übernimmt. Diese Architektur wird als Mehrschichtenarchitektur (oder auch *N-Layer*- bzw. *N-Tier*-Architektur) bezeichnet.

4.1 Rechensysteme

Das Hauptanwendungsgebiet der Informatiksysteme bestand anfangs in der Durchführung von arithmetischen Berechnungen. Konrad Zuse hat daher auch von Rechenautomaten gesprochen. Im Bereich der eher wissenschaftlichen Informatik ist man bei diesen Begriffen geblieben. Gebräuchliche Begriffe sind Rechner oder Rechensystem [Go97].

www.cm-tm.uka.de/info1
Prof. Abeck & Info1-Team

- Der Rechner ist das technische Hilfsmittel zur Realisierung (Implementierung) von Algorithmen
  - ursprünglich vor allem arithmetische Berechnungen

- Wichtige Elemente
  - Speicher für Eingabedaten und (Zwischen-) Ergebnisse
  - Programmsteuerung für die Entscheidung über die nächsten Rechenschritte

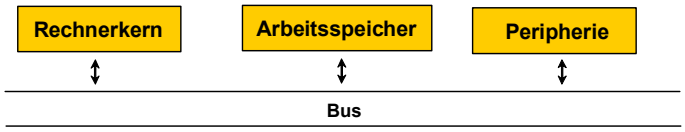
- Geschichte
  - Babbage entwickelt den ersten programmgesteuerten, mechanisch arbeitenden Rechner ('Analytische Maschine')
  - erster elektromechanisch arbeitender Rechner von Zuse ('Z3')
  - Arbeitsweise der meisten elektronisch arbeitenden Rechner geht auf Arbeiten von von-Neumann zurück

Information 31: Rechnerentwicklung

Die zwei für die Durchführung von Berechnungen notwendigen Elemente liegen auf der Hand. Man kann diese direkt aus der Überlegung ableiten, was benötigt wird, um z.B. eine per Hand durchgeführte Addition auf einer Maschine auszuführen:

- Die auf das Papier geschriebenen Zahlen müssen in der Maschine gespeichert und abgerufen werden können. Dabei müssen Eingabe-, Zwischenergebnis- und Ausgabedaten gespeichert werden können.
- Außerdem wird eine Programmsteuerung benötigt, die Konstrukte wie Bedingungen, Schleifen oder Unterprogramme auszuführen gestattet.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)



- Rechnerkern: führt das Programm aus
- Arbeitsspeicher: enthält Programme und Daten
- Peripherie: Ein-/Ausgabegeräte, (Peripherie-) Speicher
- Bus: verknüpft seriell oder parallel die oben genannten Komponenten

- Rechnerkern führt fortlaufend Befehle aus
  - z.B. Addition zweier Zahlen, boolesche Verknüpfung, Sprungbefehl
  - liest Operanden in den Speicher und schreibt Ergebnisse in den Speicher

  → Problem des "von-Neumannschen Flaschenhalses"

Information 32: Aufbau eines von-Neumann-Rechners

Die heute eingesetzten Rechensysteme entsprechen ihrem Aufbau nach einem von-Neumann-Rechner. Die Abbildung in Information 31 zeigt den Aufbau eines von-Neumann-Rechners, der auch als Princeton-Architektur bezeichnet wird. Falls mehr als ein Bus verwendet wird, spricht man von einer Harvard-Architektur.

- Herzstück der von-Neumann-Architektur ist der Rechnerkern, der auch als Zentralprozessor bezeichnet wird.
- Eine wichtige Eigenschaft des Arbeitsspeichers besteht darin, diesen nicht nur zur Speicherung von Daten, sondern auch des auszuführenden Programms zu nutzen. Das war von vornherein nicht gegeben, da die Programme zunächst auf getrennten Medien, wie z.B. Lochstreifen, gehalten wurden.
- Im Zusammenhang mit der Peripherie werden ebenfalls Prozessoren eingesetzt. Da diese hauptsächlich Kommunikationsaufgaben zu erledigen haben, werden sie als Kommunikationsprozessoren bezeichnet (im Gegensatz zu den obigen Zentralprozessoren).
- Auf dem Bus werden Befehle, Adressen, Daten, Steuersignale übertragen.

Bei der Ausführung eines Programms spielen diese vier Einheiten zusammen und bilden ein reaktives System. Es wiederholt sich dabei folgender so genannter Befehlszyklus: Der Rechnerkern holt den gerade anstehenden vom Programm vorgegebenen Befehl und führt ihn aus. Dazu sind entsprechende Speicherzugriffe erforderlich (sowohl zum Holen des Befehls, als auch zum Lesen und Schreiben der Daten).

Das Problem des so genannten von-Neumannschen Flaschenhalses resultiert aus physikalischen Gegebenheiten des (relativ schnellen) Rechnerkerns und des (relativ langsamen) Speichers.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Organisation des Speichers
  - Folge von Speicherzellen der Größe von 8 Bits (1 Byte)
  - Durchnummerieren der Speicherzellen ab 0, wobei die Nummern Adressen heißen
  - nicht nur Daten, sondern auch die Programmbefehle liegen im Speicher
- Komplexere Peripheriegeräte (z.B. Drucker, Netzwerkkarte) sind mit einem eigenen (Spezial-) Prozessor ausgestattet, der den Zentralprozessor entlastet
- Der Zugriff auf den Bus erfolgt prioritätengesteuert
- Die meisten heute eingesetzten Prozessoren sind Signalprozessoren

Information 33: Speicher und Peripherie

Der Speicher besteht aus Speicherzellen, die heute immer die Größe von 8 Bit, d.h. 1 Byte besitzen.

Daten und Befehle können länger als 1 Byte sein. In diesem Fall dient die niedrigste Adresse der beteiligten Bytes als Adresse des Datums oder des Befehls.

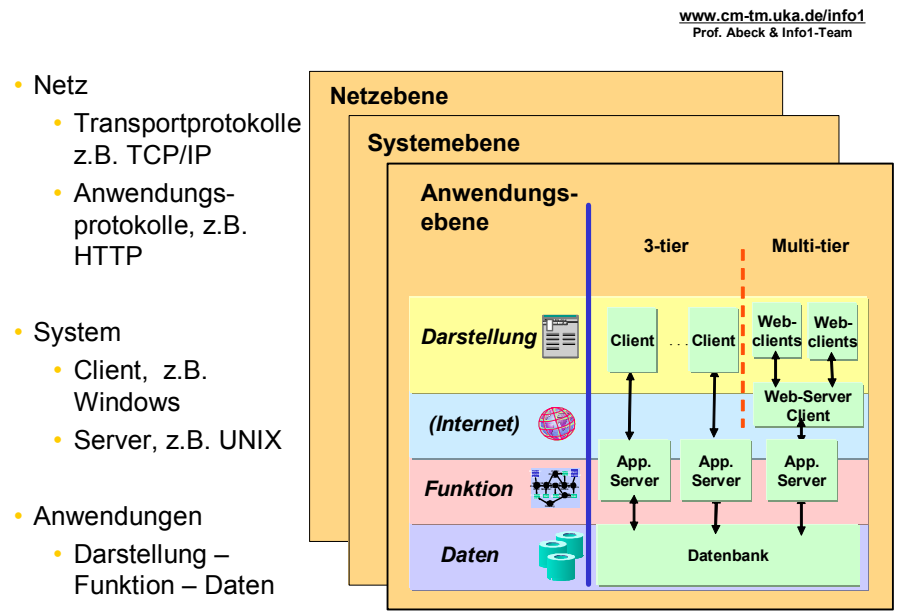
Viele Peripheriegeräte können direkt auf den Speicher zugreifen, ohne dabei den Zentralprozessor belasten zu müssen. Man spricht hierbei von einem direkten Speicherzugriff (*Direct Memory Access* DMA).

Die Prioritäten verhelfen dem Bus, bei konkurrierenden Zugriffen zu entscheiden, welcher Zugriffswunsch zuerst erfüllt wird.

Signalprozessoren verrichten Spezialaufgaben der technischen Signalverarbeitung. Neben den Prozessoren (Hardware) benötigt man die entsprechenden Programme (Software), um diese Aufgaben zu lösen.

4.2 Verteilte Systeme und Verteilte Anwendungen

Die bislang betrachtete Architektur war auf ein einzelnes Rechensystem beschränkt. Die heute eingesetzten Informatiksysteme setzen sich üblicherweise aus mehreren über ein Kommunikationsnetz verbundene Systeme zusammen. Auf dem so entstehenden Verteilten System läuft eine Anwendungssoftware, die ebenfalls ganz bestimmten Architekturkonzepten unterliegt.



Information 34: Verteiltes System

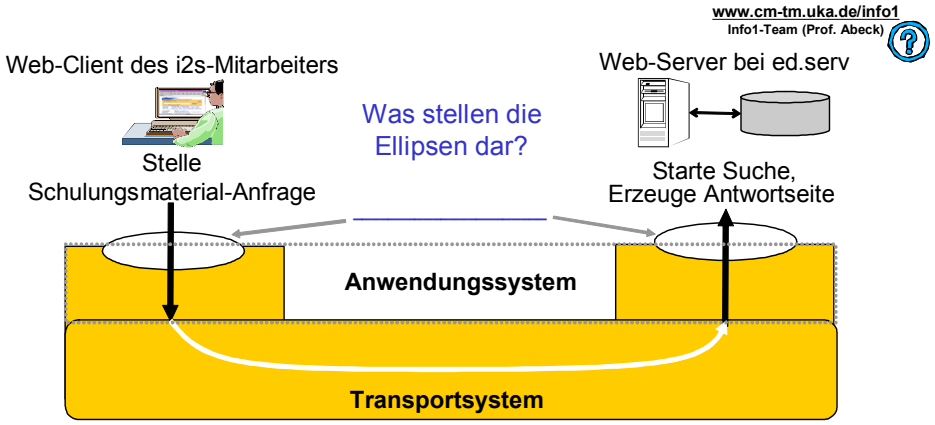
In Information 34 [C&M-AR] sind die drei Aspekte eines Verteilten Systems – Netz, (Rechen-) System, Anwendung – im Überblick dargestellt.

Heutige und zukünftige Kommunikationsnetze werden maßgeblich durch die Internet-Technologie bestimmt, deren wichtigste Bestandteile das Internet-Protokoll (IP) und die Web-Technologien [PD00] darstellen. Um die Internet-Standards herum scharen sich eine Vielzahl von konkreten Netztechnologien und -produkten, die einen bestimmten Einsatzbereich (z.B. Hochleistungs- oder Mobilkommunikation) adressieren. Hieraus resultiert eine heterogene und komplexe Netzinfrastruktur mit unterschiedlichen Eigenschaften. Von den über diese Netzinfrastruktur angeschlossenen Rechensystemen und den darauf laufenden (verteilten) Anwendungen wird ein Kommunikationsdienst mit bestimmten, möglichst garantierten Qualitätseigenschaften gefordert (so genannter *Quality of Service*, QoS).

Die Rechensysteme [CD01] arbeiten auf einer Client-Server-Basis, wobei im Client-Bereich die PCs mit dem Windows-Betriebssystem vorherrschen und im Server-Bereich zudem verstärkt Varianten des Unix-Betriebssystems (z.B. Linux, SUN Solaris, HP UX, IBM AIX) und Großrechner-Betriebssysteme zum Einsatz kommen. Da einige der Großrechner-Betriebssysteme eher eine "Altlast" darstellen, werden sie auch als *Legacy*-Systeme bezeichnet.

Die Verteilten Anwendungen [In02] laufen auf dieser heterogenen (d.h. eine Vielzahl von Betriebssystemen einbeziehenden) vernetzten Systemlandschaft. In Verteilten Anwendungen lassen sich drei elementare Aspekte der Daten, der Funktionen und der Darstellung unterscheiden.

Die Struktur eines Kommunikationsnetzes wird anhand des Beispielszenarios in Interaktion 35 skizziert.

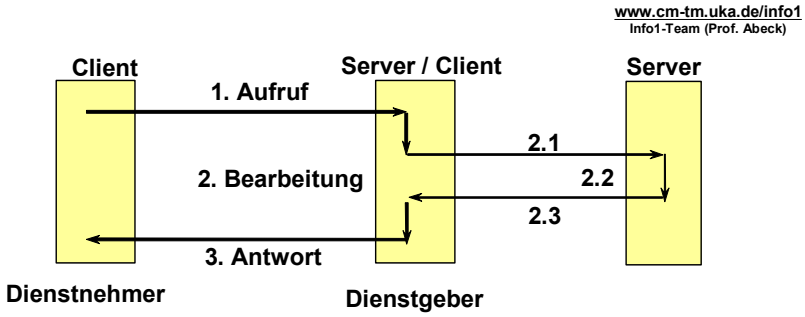


- Transportsystem stellt eine sichere Ende-zu-Ende-Verbindung her
  - wird aus Vermittlungsknoten (im Internet so genannte IP-Router) gebildet
- Anwendungssystem stellt anwendungsnahe Kommunikationsdienste bereit
  - im Beispiel ist der Dienst der Zugriff auf Web-Seiten

Interaktion 35: Nutzung des Kommunikationsnetzes im Beispielszenario

Wie die Abbildung zeigt, setzt sich das Kommunikationssystem aus zwei Teilsystemen zusammen. Während das Transportsystem den sicheren Transport der Daten zwischen den beiden kommunizierenden Rechensystemen (den Endsystemen) übernimmt, stellt das auf dem Transportsystem aufsetzende Anwendungssystem höherwertige Dienste bereit, wie beispielsweise der Zugriff auf Web-Seiten, der in Interaktion 35 gezeigt ist.

Im Beispiel zur Nutzung des Kommunikationsnetzes waren die zwei kommunizierenden Systeme ein Web-Client und ein Web-Server. Information 36 stellt das allgemeine Prinzip von Client-Server dar.



- Unterbeauftragung: Server beauftragt im Rahmen der Bearbeitung eines Auftrags einen weiteren Server

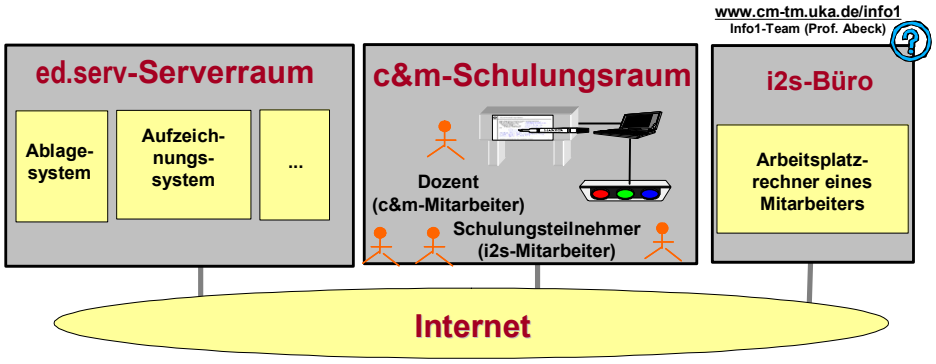
- Ein Beispiel eines wichtigen Client-Server-Basisprotokolls ist der Remote Procedure Call (RPC)
  - realisiert einen entfernten Prozeduraufruf

Information 36: Client-Server-Prinzip

Das in Information 36 gezeigte Client-Server-Prinzip dient zur allgemeinen Strukturierung der Kooperationsbeziehung auf der System-Ebene. Das Prinzip geht davon aus, dass der Kooperation von zwei Systemen eine Dienstbeziehung zugrunde liegt. Dabei übernimmt das eine System die Rolle des Dienstnehmers – genannt Client – und das andere System die Rolle des Diensterbringers – genannt Server.

Ein System kann in verschiedenen Dienstbeziehungen stehen und in einer Beziehung die Rolle des Clients und in einer anderen die des Servers übernehmen. Ein Spezialfall eines solchen Rollentausches ist die Unterbeauftragung, in dem ein Server zur Bearbeitung eines Dienstes ein anderes System als Server in Anspruch nimmt und hierzu die Rolle des Clients einnimmt.
Es besteht ein enger Zusammenhang zwischen dem Client-Server-Prinzip und den Protokollen auf der Netzebene. So liegt dem zuvor in Interaktion 35 skizzierten Zugriff auf einen Web-Server das Client-Server-Prinzip zugrunde. Das Beispiel kann problemlos dahingehend erweitert werden, dass der vom Web-Client angefragte Web-Server zur Erbringung der Anfrage andere Server-Systeme in Anspruch nehmen muss.

Aus der Sicht der verteilten Anwendung muss ein Protokoll vorhanden sein, das das Client-Server-Prinzip mit der Übertragung des Aufrufs und der Antwort flexibel unterstützt. Ein solches Protokoll ist der *Remote Procedure Call* (RPC), der in unterschiedlichen Ausprägungen in verschiedenen standardisierten Verteilten Systemarchitekturen auftritt.



- Welche Systeme übernehmen die Rolle eines Servers bzw. eines Client?
  Server-Systeme: _____
  Client-Systeme: _____

- Zwischen welchen dieser Systeme könnte ein RPC ablaufen und was wäre ein Beispiel eines entfernten Aufrufs?
  _____

Interaktion 37: Client-Systeme und Server-Systeme im Beispielszenario

In der abschließenden Interaktion 37 ist zu ermitteln, an welchen Stellen im Beispielszenario Client-Systeme und Server-Systeme auftreten und es sind Beispiele für das Zusammenspiel dieser Systeme auf der Grundlage des *Remote Procedure Calls* anzugeben.

Mit den Begriffen der Information, des Algorithmus, des Modells und der Architektur sowie dem in der Kurseinheit EINFÜHRUNG IN DAS FACHGEBIET dargestellten Systembegriff wurde der Kern der Informatik dargestellt.

VERZEICHNISSE

Abkürzungen und Glossar

Abkürzung oder Begriff	**Langbezeichnung und/oder Begriffserklärung**
Algorithmus	Ein Verfahren mit einer präzisen (d.h. in einer eindeutigen Sprache abgefassten) endlichen Beschreibung unter Verwendung effektiver (im Sinne von tatsächlich ausführbarer) Verarbeitungsschritte [Br98].
Architektur	Spezifikation der grundlegenden Struktur eines Systems.
Client-Server	Allgemeines Architekturprinzip zur Strukturierung von Kooperationsbeziehungen auf der System-Ebene.
Codierung	Festlegung der Darstellung von Informationen in Form von Zeichen auf der Grundlage eines Codes.
c&m	cooperation & more Im Szenario des Internet-basierten Wissenstransfers auftretendes Schulungsunternehmen.
DMA	*Direct Memory Access* Direkter Zugriff eines Peripheriegeräts auf den Speicher, ohne dabei den Zentralprozessor zu belasten.
Euklidscher Algorithmus	Der älteste und damit erste Algorithmus, der den größten gemeinsamen Teiler von zwei ganzen Zahlen berechnet.
ed.serv	eudcational.services Name des IT-Dienstleisters, der im Szenario des Internet-basierten Wissenstransfers auftritt.
Legacy-Systeme	Systeme, die auf veralteten Technologien basieren aber aus Kostengründen nicht erneuert werden können bzw. sollen. Wörtliche Übersetzung von *Legacy*: Altlast
i2s	intellligent internet solutions Name des als Kunden einer Schulung auftretenden Unternehmens, das im Szenario des Internet-basierten Wissenstransfers auftritt.
Mehrschichtenarchitektur	Architektur einer verteilten Anwendung, in der die Aspekte der Anwendung in mehrere Schichten (insbes. Darstellung, Verarbeitung, Daten) aufgeteilt wird. Englisch: *N-Layer Architecture*
Modell	Abbildung und Darstellung eines Ausschnitts der Wirklichkeit.
QoS	*Quality of Service*

	Die von einem IT-Dienst (z.B. Kommunikationsdienst) garantierten Qualitätseigenschaften, wie z.B. maximale Antwortzeit oder minimale Verfügbarkeit.
Semiotik	Lehre der Zeichen, die sich aus der Syntax, der Semantik und der Pragmatik zusammensetzt.
Signal	Darstellung einer Mitteilung durch die zeitliche Veränderung einer physikalischen Größe.
UML	*Unified Modeling Language* (vereinheitlichte Modellierungssprache) Sprache, die aus graphischen Elementen besteht und zur semi-formalen Beschreibung von beliebigen Gegenständen (z.B. Software-Systeme oder Geschäftsbereiche) genutzt wird.
von-Neumannscher Flaschenhals	Bezeichnung eines Engpasses, der in der von-Neumann-Architektur aufgrund der im Vergleich zur Prozessorbearbeitungszeit relativ langen Speicherzugriffszeiten entsteht. Synonymer Begriff: von-Neumann-Engpaß
von-Neumann-Rechner	Ein aus Rechnerkern, Arbeitsspeicher, Peripherie und einem diese Einheiten verbindenden Bus bestehender Rechner.

Index

Informationen und Interaktionen

Literatur

[AL+03] Sebastian Abeck, Peter C. Lockemann, Jochen Schiller, Jochen Seitz: Verteilte Informationssysteme: Integration von Datenübertragungstechnik und Datenbanktechnik: dpunkt.verlag, 2003.

[BB93] Gilles Brassard, Paul Bratley: Algorithmik – Theorie und Praxis, Wolfram's Verlag, 1993.

[BG73] Friedrich L. Bauer, Gerhard Goos: Informatik – Eine einführende Übersicht – Erster Teil, Springer-Verlag, 1973.

[Br98] Manfred Broy: Informatik – Eine grundlegende Einführung, Band 1: Programmierung und Rechstrukturen, Springer Verlag 1998.

[BH03] Bernd Brügge, Christian Herzog: Einführung in die INFORMATIK I, Vorlesungsfolien, TU München, 2003.

[C&M-AR] Cooperation&Management, ARCHITEKTUR UND RAHMENWERKE, Kursdokument zur Vorlesung "ENTWICKLUNG VON INTERNET-SYSTEMEN UND WEB-APPLIKATIONEN", http://www.cm-tm.uka.de/iswa, Universität Karlsruhe (TH), C&M (Prof. Abeck)

[C&M-BS] Cooperation&Management: BITÜBERTRAGUNG UND SICHERUNG, Kursdokument zur Vorlesung "KOMMUNIKATION UND DATENHALTUNG", http://www.cm-tm.uka.de/kud, Universität Karlsruhe (TH), C&M (Prof. Abeck)

[Go97] Gerhard Goos: Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer-Verlag 1997.

[PD00] L. L. Peterson, B. S. Davie, Computernetze - Ein modernes Lehrbuch: dpunkt.verlag, 2000.

ALGEBRAISCHE STRUKTUREN UND FORMALE SYSTEME

Kurzbeschreibung

Vorliegende Kurseinheit behandelt mit den Relationen (Halbgruppen, Graphen) und den Algebren (Formeln, Boolesche Algebra) zwei zentrale mathematische Hilfsmittel der Informatik. Die algebraischen Strukturen dienen u.a. zur Entwicklung von formalen Systemen und von Textersetzungssprachen. Durch Chomsky-Grammatiken werden die Sprachen in insgesamt vier Sprachklassen eingeordnet werden. Mit den Endlichen Automaten wird ein Maschinenmodell vorgestellt, das die einfachste Sprachklasse in der Chomsky-Hierarchie zu verarbeiten gestattet.

Schlüsselwörter

Algebraische Struktur, Halbgruppe, Monoid, Relation, Graph, Warshall-Algorithmus, Formel, Term, Boolesche Algebra, Dualitätsprinzip, Normalform, Algorithmus, Textersetzung, Semi-Thue-System, Grammatik, Chomsky-Sprachklassen, endlicher Automat, regulärer Ausdruck

Lernziele

1. Die Bedeutung von algebraischen Strukturen in der Informatik wird erkannt.
2. Der vielseitige Einsatz von Relationen bzw. Graphen zur formalen Beschreibung von Sachverhallten und die Möglichkeiten der darauf aufsetzenden (Graph-) Algorithmen zur Lösung von Problemstellungen werden verstanden.
3. Der Aufbau der booleschen Algebra kann wiedergegeben werden.
4. Die Bedeutung der Textersetzung als elementarste Form der Beschreibung von Algorithmen und der Verarbeitung von Informationen wird verstanden.
5. Ein endlicher Automat zur Erkennung und Erzeugung von Wörtern einer formalen Sprache kann erstellt werden.
6. Die durch Textersetzungssysteme (Semi-Thue-Systeme) und endliche Automaten erzeugten Sprachen können in das durch die Chomsky-Sprachklassen beschriebene Spektrum der Formalen Sprachen eingeordnet werden.

Hauptquellen

- Gerhard Goos: Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer Verlag 1997.

Inhaltsverzeichnis

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- RELATIONEN
  - Halbgruppen und Monoide, Operationen und Gesetze, algebraische Abgeschlossenheit, Einselement, Graphen, gerichtete und ungerichtete Graphen, Repräsentation von Graphen, Warshall-Algorithmus
- ALGEBREN
  - Formeln, Term, Kantorowitsch-Baum, Boolesche Algebra, Gesetze, Dualität
- SEMI-THUE-SYSTEME
  - Regeln und Metaregeln, Markov-Algorithmen, Formale Systeme
- GRAMMATIKEN
  - Nichtterminale und Terminale, Axiom, Chomsky-Hierarchie, Ableitungsbaum
- ENDLICHE AUTOMATEN
  - Mealy-Automat, Moore-Automat, Akzeptor, Reguläre Ausdrücke, Einordnung in die Chomsky-Hierarchie

Information 1: ALGEBRAISCHE STRUKTUREN UND FORMALE SYSTEME

1 RELATIONEN

Relationen stellen Zusammenhänge zwischen Elementen einer oder mehrerer Mengen dar. In praktisch jeder Aussage werden gewisse Gegenstände in Beziehung gesetzt, also Relationen gebildet [Go97].

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Durch Relationen werden Elemente einer oder mehrerer Mengen in Beziehung gesetzt
  - praktisch jede Aussage enthält Relationen
  - Beispiel: "Das Haus hat vier Außenwände"
- In der Informatik werden Relationen zur Modellierung von Systemen benötigt
  - Relationen sind ein wesentlicher Bestandteil der verschiedenen Diagramme der Unified Modeling Language
- Aus der graphischen Darstellung von Relationen resultieren die Graphen

Information 2: RELATIONEN - Überblick

Beispielsweise werden in der Aussage Das Haus hat vier Außenwände (siehe Information 2) die Gegenstände Haus und Außenwände in Relation gesetzt. Die zu einem Element aufstellbaren Relationen können sehr vielfältig sein. Eine andere Aussage zum Gegenstand Haus ist beispielsweise Das Haus hat vier Buchstaben. Hierdurch wird eine ganz andere Beziehung zu ein und demselben Gegenstand Haus aufgebaut, die nicht auf den Begriff, sondern auf das Datum abzielt.

Relationen sind in der Informatik so wichtig, weil sie eine ganz wesentliche Grundlage zur Modellierung von Systemen bilden. Durch die in der Kurseinheit GRUNDBEGRIFFE DER INFORMATIK eingeführten Diagrammtypen der *Unified Modeling Language* (UML) werden Relationen dargestellt.

In diesem Kapitel werden mit den Halbgruppen und Graphen die konzeptionellen Grundlagen zu den Relationen gelegt.

1.1 Halbgruppen und Monoide

Im Folgenden werden ganz elementare mathematische Begriffe anhand eines einfachen Sachverhalts, nämlich dem Nebeneinanderschreiben von Zeichen, eingeführt. Das Ziel besteht darin, die dadurch festgelegte Relation mathematisch geeignet zu formulieren.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Die in der Zeichenverarbeitung auftretende Beziehung des Nebeneinanderschreibens von Zeichen lässt sich als eine Halbgruppe auffassen

- Mathematische Formulierung als Algebra
  1. Menge
     - Zeichenreihen über der Grundmenge $\sum^+$
  2. Operationen
     - $\cdot$ Konkatenation
  3. Gesetze
     - HG1: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ Assoziativgesetz

- Die Eigenschaft der algebraischen Abgeschlossenheit ist erfüllt

- Die beschriebene Algebra ($\sum^+$, $\cdot$, HG1), kurz auch als $\sum^+$ bezeichnet, bildet eine Halbgruppe

Information 3: Halbgruppe

Die in Information 3 angestellten Überlegungen führen zum zentralen Begriff der Algebra. Eine Algebra ist definiert als ein Tripel bestehend aus (Trägermengen, Operationen, Gesetze).

(1) Trägermenge
Im Beispiel wird die Trägermenge gebildet durch die Zeichen bzw. genauer durch die Folgen von Zeichen, also die Zeichenreihen.

(2) Operationen
Die einzige Operation im Beispiel ist das Nebeneinanderschreiben, die auch als Konkatenation oder Verkettung bezeichnet wird. Hierzu wird durch das Zeichen '$\cdot$' ein geeignetes mathematisches Symbol eingeführt.

(3) Gesetze
Die Gesetze verbinden Trägermenge und Operationen, da durch diese die Frage beantwortet wird, welche Aussagen über die Operation, die auf der Trägermenge arbeitet, getroffen werden können. Ein offensichtlich geltendes Gesetz, die die Konkatenation erfüllt, ist das Assoziativgesetz.

Neben den Gesetzen wird von jeder Algebra die Eigenschaft der algebraischen Abgeschlossenheit gefordert, die eine Algebra zu erfüllen hat.

Algebren, die bestimme Strukturen haben, erhalten einen Namen. So heißt eine Algebra, die im Zusammenhang mit der Konkatenation von Zeichen erstellt wurde, eine Halbgruppe. Eine Halbgruppe ist also eine spezielle Algebra

$$\mathcal{A} = (\Sigma^+,\{\cdot\},\{HG1\})$$

Eng mit dem Begriff der Halbgruppe ist der Begriff des Monoids verknüpft, der in Interaktion 4 näher ausgeführt wird.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ein Monoid ist eine Halbgruppe mit Einselement
  - Eigenschaft eines Einselements ε
    (1) $\varepsilon \cdot a = a$
    (2) $a = a \cdot \varepsilon$
  - das Einselement ist eindeutig

  Beweis: _____

  _____

- Beispiel eines Monoids: $\Sigma^* := \Sigma^+ \cup \{\varepsilon\}$
  - ε ist das leere Wort, d.h. die Länge $|\varepsilon|$ ist 0
  - falls w^n die n-fache Wiederholung des Wortes w bezeichnet, so gilt $w^0 = \varepsilon$

- Satz
  Das Monoid Σ^* über einem endlichen Zeichenvorrat Σ ist abzählbar
  - Beweis mittels Gödelnummerierung

Interaktion 4: Monoid

Das in einem Monoid geforderte Einselement wird auch als neutrales Element bezeichnet. In einer Halbgruppe kann es Elemente geben, die nur eine der beiden in Interaktion 4 angegebenen Eigenschaften eines Einslelements erfüllen. Falls nur Eigenschaft (1) erfüllt ist, so heißt das Element Linkseinselement, falls nur (2) erfüllt ist, entsprechend Rechtseinselement. Ein Element, das die Eigenschaften (1) und (2) erfüllt, ist eindeutig, wie in der Interaktion zu zeigen ist. In der Halbgruppe A = (Σ+,{·},{HG1}) ist offensichtlich das leere Wort ε das Einselement in der Algebra.

Die Informatik behandelt vorwiegend solche Strukturen, die auf einem Rechensystem bearbeitet werden können. Hieraus resultiert die Eigenschaft, dass die Zeichenvorräte zwangsläufig endlich sein müssen. Die Konsequenz, die sich für Monoide über einem endlichen Zeichenvorrat ergibt, ist die Abzählbarkeit. Der Beweis der Abzählbarkeit erfolgt mittels der so genannten Gödelnummerierung (Kurt Gödel, Logiker des 20. Jahrhunderts).

Die algebraische Struktur der Monoide ist für die Informatik bedeutsam, da sie zum Aufbau von listen- und sequenzartigen Datenstrukturen dient (siehe Interaktion 5), die von zahlreichen Algorithmen genutzt werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- In der Informatik werden Monoide U^* häufig über anderen Grundmengen als Zeichen gebildet
  - Grundmenge U kann aus Zeichenreihen bestehen, die aus einem anderen Monoid $\sum^*$ stammen
  - Elemente von U* heißen Listen
    - oder alternativ: Sequenzen, Folgen
  - Notation:
    - Liste: $[x_1,x_2,...,x_n]$, wobei $x_i \in U$
    - [] ist die leere Liste
    - [U] als alternative Schreibweise für U^*

- Beispiel: U = {Apfel,Birne,Pflaume,Kirsche,Traube}
  - [Apfel,Birne], [Pflaume,Kirsche,Traube] sind Listen
  - append([Apfel,Birne], [Pflaume,Kirsche,Traube]) ist eine alternative Funktionsschreibweise für

Interaktion 5: Monoid U* über Monoid Σ*

Listen gehen aus Monoiden durch eine geeignete Abänderung der Trägermengen hervor. Die Trägermengen eines Listen-Monoids Monoid U* sind keine einzelnen Zeichen, sondern bereits Zeichenreihen aus einem Monoid Σ^*.

Die in Interaktion 5 angegebene Listennotation für eine beliebige Liste, die leere Liste und den ganzen Listen-Monoid vereinfacht die Aufschreibung, macht die Syntax intuitiver und damit besser lesbar. Die Anwendung der Notation soll das Beispiel verdeutlichen, dessen Trägermenge U aus den fünf angegebenen Texten – in diesem Fall verschiedene Sorten Obst – besteht.

Eine weitere Art von Notation ist die Funktionsschreibweise, in der das mathematische Verknüpfungssymbol gegen eine Funktion ersetzt wird. Die Funktionsschreibweise ist eine Form der so genannten Präfixnotation.

1.2 Graphen

Monoid-Operationen stellen die elementarste Relation zwischen Elementen einer Trägermenge dar. Im Folgenden werden die Beziehungen in Form des Begriffs der Relation mathematisch präzisiert und mittels Graphen veranschaulicht.

- Relationen
  - Grundmengen U, V
  - Notation: $\rho \subseteq U \times V$
- Jede Relation kann als Teilmenge des kartesischen Produkts $(U \cup V) \times (U \cup V)$ angesehen werden
- Homogene Relation
  - nur eine Grundmenge E, d.h. $\rho \subseteq E \times E$
- Die Informatik interessiert sich speziell für homogene Relationen mit einer endlichen Grundmenge E
  - Durchnummerieren der Elemente e_i mit i = 0, ..., n - 1
  - gerichtete Graphen ermöglichen eine anschauliche Darstellung

Information 6: Relationen und Graphen

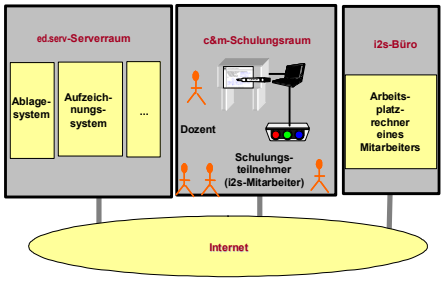
Eine Relation liefert den grundlegenden Formalismus, Beziehungen zwischen Gegenständen zweier Grundmengen U und V zu beschreiben. Die in der Informatik übliche Beschränkung auf homogene Relationen ist bei näherer Betrachtung keine echte Einschränkung. Wie in Information 6 ausgeführt ist, lassen sich nicht-homogene Relationen $U \times V$ durch die Vereinigung der beiden Mengen U und V sowie die Bildung des kartesischen Produkts auf homogene Relationen zurückführen. Die nicht-homogene Relation ist dann eine Teilmenge der konstruierten homogenen Relation. Eine homogene Relation (bezeichnet mit ρ) ist also auf nur einer Grundmenge definiert.

Die Informatik stellt neben der Homogenität noch eine weitere Forderung an die Grundmenge, nämlich die Eigenschaft der Endlichkeit. Das ermöglicht dann ein Durchnummerieren der Elemente und eine anschauliche Darstellung der Relation in Form von gerichteten Graphen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ein gerichteter Graph G ist ein Tupel (E,K) mit
  - Grundmenge E = $\{e_i\}$ eine Menge von Ecken
  - Relation $K \subseteq E \times E$ eine Menge von Kanten
    - alternative Notationen für Kanten
      - $(e,e') \in K$ oder $e \rightarrow_G e'$
      - $e \rightarrow e'$

- Der Inhalt des Web-basierten Ablagesystems aus dem Beispielszenario kann als gerichteter Graph aufgefasst werden
  - K: _____
  - E: _____



Interaktion 7: Gerichteter Graph

Ein gerichteter Graph setzt sich aus zwei Mengen zusammen, einer Menge von Ecken (Knoten) und einer Menge von Kanten. Die Kantenmenge K stellen dabei zweistellige Relationen auf der Grundmenge E der Ecken dar.

Relationen und Graphen spielen in der Informatik eine zentrale Rolle, da zahlreiche Problemstellungen auf Graphen und entsprechende Graphenalgorithmen zurückgeführt werden können. Die Begriffe ‘Relation’ und ‘Graph’ werden dabei weitgehend synonym verwendet.

In Interaktion 7 soll ein mögliches Einsatzfeld von Graphen anhand des in der Kurseinheit EINFÜHRUNG IN DAS FACHGEBIET eingeführten Beispielszenarios und des darin enthaltenen Web-basierten Ablagesystems skizziert werden.

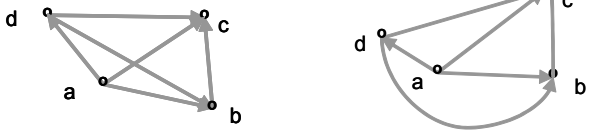
www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ein Graph heißt endlich, wenn E endlich ist

- Graphische Darstellung von
  - Ecken als Punkte
  - Kanten als Pfeile

- Die Positionierung der Ecken und Kanten ist beliebig

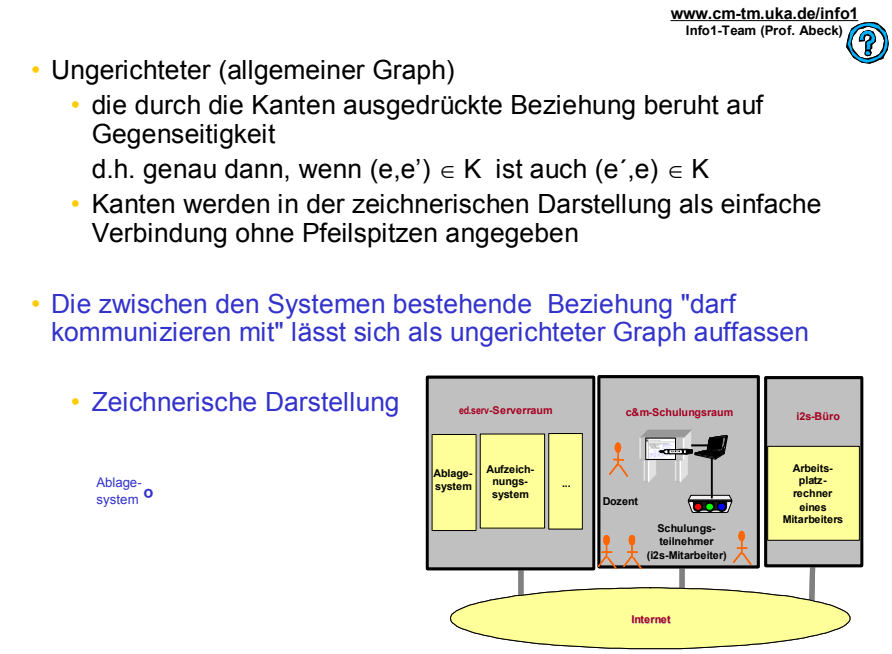
- Beispiel: Zwei mögliche Darstellungen des Graphen {(a,b),(a,c),(a,d),(b,c),(d,b),(d,c)}



Information 8: Graphische Darstellung eines endlichen gerichteten Graphen

Die endlichen Graphen – also Graphen mit endlicher Eckenzahl – lassen sich zeichnen, wie das Beispiel in Information 8 zeigt. Durch das Beispiel wird zudem gezeigt, dass es unterschiedliche Darstellungen von ein und demselben Graphen gibt. Freiheitsgrade bei der Graphen-Darstellung sind

1. eine beliebige Positionierung der Ecken und
2. ein beliebiger Linienzug der Kanten.



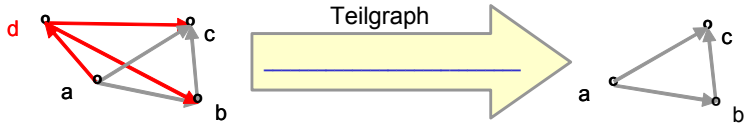
Interaktion 9: Ungerichteter Graph

Es gibt Sachverhalte, bei denen eine Relation von a nach b zwangsläufig die Relation b nach a nach sich zieht. Eine derartige auf Gegenseitigkeit beruhende Relation ist beispielsweise ist über einen Wanderweg verbunden mit. Wanderwege sind üblicherweise in beide Richtungen benutzbar. Zeichnerisch ist ein ungerichteter Graph daran zu erkennen, dass die Kanten keine Pfeilspitzen aufweisen.

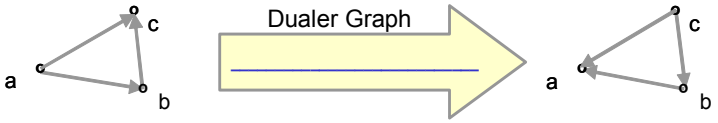
Ein weiteres Beispiel eines ungerichteten Graphen, der aus dem Beispielszenario gegriffen wurde, findet sich in Interaktion 9. Die zeichnerische Darstellung dieser Relation ist als Aufgabe im Rahmen der Interaktion anzugeben.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Teilgraph G´= (E´,K´) zum Graphen G = (E,K)
  - Einschränkung der Relation ρ auf eine Teilmenge E´⊆ E; Notation: $\rho|_{E'}$
  - K´ ={(e,e´) | e,e´ ∈ E´ und (e,e´) ∈ K}



- Dualer Graph G^T = (E,K^T)
  - (e,e´) ∈ K^T genau dann, wenn (e´,e) ∈ K
  - Der duale Graph G^T entsteht aus G, wenn alle Richtungen der Pfeile in G umgekehrt werden



Interaktion 10: Teilgraph und dualer Graph

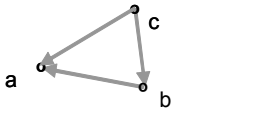
Ein Teilgraph G´ geht aus einem Graphen G durch Einschränkung der Relation auf gewisse Ecken aus E hervor. Zu K´ gehören alle Kanten aus K, die Ecken e, e´∈E´ miteinander verbinden.

Eine weitere Graphenart, die aus einem gegebenen Graphen G hervorgeht, ist der als G^T bezeichnete duale Graph. Die Menge der Ecken E bleibt beim dualen Graphen unverändert und die Menge der Kanten K^T geht durch Vertauschen der Start- und Zielecke jeder Kante hervor.

In Interaktion 10 soll der Übergang des Ausgangsgraphen in einen Teilgraphen bzw. in einen dualen Graphen durch Verwendung der eingeführten Notationen beschrieben werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ausgangsmenge e• und Eingangsmenge •e
  - Menge der Kanten, die bzgl. der Ecke e aus- bzw. eingehen
  - |e•| heißt Ausgangsgrad, |•e| heißt Eingangsgrad
  - in einem ungerichteten Graphen gilt |e•| = |•e| = grad(e) und heißt Grad der Ecke e



a• = _____ |a•| = ___

•a = _____ |•a| = ___

- Für beliebige Graphen gilt: $\sum_{e \in E} |e^{\bullet}| = \sum_{e \in E} |{}^{\bullet}e|$
- Vollständiger (ungerichteter) Graph
  - zwischen je zwei Ecken existiert eine (ungerichtete) Kante

Interaktion 11: Grad einer Ecke

Die Eingangs- und Ausgangsmenge beinhalten die von einer Ecke ausgehenden bzw. in eine Ecke mündenden Kanten. Zur Angabe dieser Mengen wird eine einfache Punkt-Notation eingeführt, wie in Interaktion 11 ausgeführt ist.

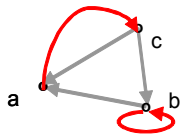
Eng verknüpft mit dem Eingangs- und Ausgangsmengen ist der so genannte Grad. Der Grad gibt die Anzahl der Kanten an, die von der Ecke ausgehen bzw. zu dieser Ecke führen. Entsprechend dieser beiden Fälle werden ein Ausgangsgrad und ein Eingangsgrad unterschieden. Zur Angabe dieser Grade wird die Punktnotation um Betragsstriche ergänzt.

Die in Interaktion 11 als Formel beschriebene Gleichheit der Summe der Ausgangs- und Eingangsgrade aller Ecken eines Graphen lässt sich einfach erklären. Für einen gerichteten Graphen gilt zudem, dass die Eingangs- bzw. Ausgangsgradsumme genau der Anzahl der Kanten im Graphen entspricht.

Ein vollständiger Graph hat die Eigenschaft, dass jede Ecke mit jeder anderen im Graphen auftretenden Ecke verbunden ist. In einem vollständigen ungerichteten Graph gilt für alle Ecken e, dass grad(e) = |E| - 1. Diese Eigenschaft ist gleichbedeutend mit der Eigenschaft, dass jede Ecke mit jeder übrigen Ecke verbunden ist (d.h. $\forall e, e' \in E: (e, e') \in K$).

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Folge von e_0 nach e_n über $e_1,....,e_{n-1}$
  - heißt Weg $e_0 \rightarrow^* e_n$ der Länge n
  - e_n heißt dann von e_0 erreichbar
- Zyklus (Kreis)
  - ein Weg $e_0 \rightarrow^* e_n$ mit $n \geq 1$ und $e_0 = e_n$



- Einfacher Zyklus
  - alle Ecken e_i sind verschieden
- Hamiltionscher Kreis
  - jede Ecke des (ungerichteten) Graphen ist genau einmal enthalten
- Eulerscher Zyklus
  - jede Kante ist genau einmal enthalten

Information 12: Weg und Zyklus

Kanten können miteinander verknüpft werden, so dass ein so genannter Weg entsteht. Hierzu verbindet man solche zwei Kanten, bei denen die Eingangsecke der einen Kante mit der Ausgangsecke der anderen Kante übereinstimmen. Bei ungerichteten Graphen sind zwei Kanten dann verbunden, wenn sie eine gemeinsame Ecke besitzen.

Wege werden durch die in Information 12 eingeführte Pfeilnotation beschrieben. Durch die Notation $e^0 \rightarrow^+ e^n$ wird von einem Weg eine Mindestlänge von 1 gefordert, d.h. Wege der Länge 0 sind in diesem Fall ausgeschlossen.

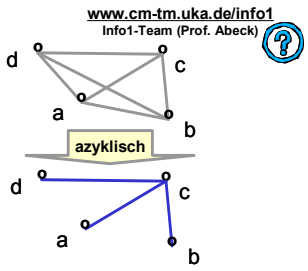
Ein spezieller Weg ist der Zyklus oder Kreis, der bei der Ecke (e_n) endet, bei der er beginnt (e_0). Ein Zyklus der Länge 1 wird als Schlinge bezeichnet (d.h. ein Knoten, der Anfangs- und Endpunkt ist).

Die bislang behandelten Graphen enthalten keine Schlingen und auch keinen sonstigen längeren Zyklus. In dem in Information 12 dargestellten Beispielgraphen bestehen die Zyklen $a \to c \to a$ und $a \to c \to b \to a$ sowie eine Schlinge $b \to b$.

Es werden verschiedene Arten von Zyklen bzw. Kreise unterschieden:
- Ein einfacher Zyklus ist gegeben, wenn keine zwei gleichen Ecken auftreten.
- Ein hamiltonscher Kreis ergibt sich durch eine Verschärfung dieser Forderung, nämlich dass jede Ecke nicht nur höchstens einmal, sondern genau einmal auftreten darf.
- Der eulersche Zyklus stellt die Anforderung, dass jede Kante des betrachteten Graphen genau einmal enthalten sein muss.

Der eulersche Zyklus wurde von Leonhard Euler (schweizerischer Mathematiker des 18. Jh.) im Zusammenhang mit dem Königsberger Brückenproblem definiert, bei dem ein Weg über die damaligen sieben Brücken der Pregel gesucht wurde, bei dem keine der Brücken zweimal überquert werden musste. Bei der Darstellung als Graph sind die Brücken die Kanten und das die Brücken verbindende Land die Ecken.

- Azyklischer Graph: Graph ohne Zyklen
- Wald: Azyklischer ungerichteter Graph
- Ungerichteter Baum: Wald, bei dem je zwei Ecken durch genau einen Weg verbunden sind



Welcher spezielle Graph liegt hier vor?

- Satz: Für einen ungerichteten Baum G = (E,K) mit endlich vielen Ecken gilt |E| = |K| + 1
  - Beweis durch vollständige Induktion entlang der Eckenzahl

Interaktion 13: Spezielle Graphen im Zusammenhang mit Zyklen

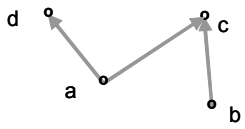
Im Zusammenhang mit Zyklen werden spezielle Graphen unterschieden, wie z.B. die in der Informatik besonders wichtigen Bäume. Die wesentlichen Eigenschaften dieser Graphenart sind die Zyklenfreiheit und das Verbundensein von je zwei Ecken über beliebige Wege (so genannter zusammenhängender Graph).

Erfüllt ein ungerichteter Graph nur die Eigenschaft der Zyklenfreiheit, nicht aber die Eigenschaft des Zusammenhängens, so spricht man von einem Wald. Offensichtlich besteht ein Waldgraph aus einzelnen Baumgraphen.

Bei einem ungerichteten Baum gilt für die Ecken- und Kantenanzahl, dass es genau eine Ecke mehr als Kanten gibt. Diese Eigenschaft ist in Interaktion 13 durch einen Induktionsbeweis zu zeigen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Gerichteter Wald
  Ein azyklischer Graph, bei dem alle Knoten einen Eingangsgrad $|\bullet e| \leq 1$ besitzen
- Gerichteter Baum
  Ein gerichteter Wald, bei dem es genau eine Ecke e mit $|\bullet e| = 0$ gibt
  - diese Ecke heißt die Wurzel des Baumes
- Blatt
  - bei einem ungerichteten Graphen: Ecke mit grad(e) = 1
  - bei einem gerichteten Graphen: Ecke mit Ausgangsgrad $|e\bullet| = 0$



Der Graph ist (1) in einen gerichteten Wald und anschließend (2) in einen gerichteten Baum zu überführen

(1) _____

(2) _____

Welche Ecken des erstellten gerichteten Baumes sind Blätter?

_____

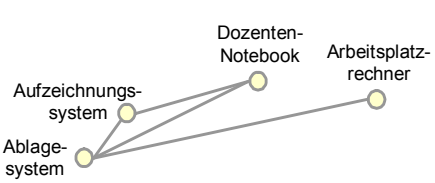
Interaktion 14: Gerichtete Wälder und Bäume

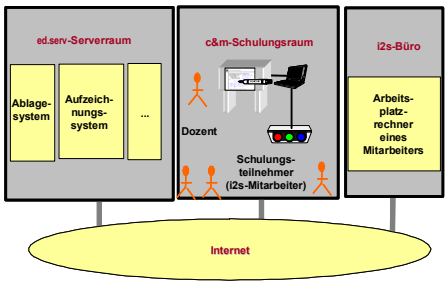
Die Angaben in Interaktion 13 haben sich auf ungerichtete Graphen bezogen. Im Falle von gerichteten Graphen gelten für einen Wald und einen Baum die in Interaktion 14 beschriebenen Eigenschaften, die gewisse Forderungen an den Eingangsgrad der im Graphen auftretenden Ecken haben.
Von einem gerichteten Wald wird gefordert, dass zu jedem Knoten höchstens eine Kante führt. Offensichtlich ist das für den angegebenen Beispielgraphen nicht erfüllt. Ein gerichteter Baum muss zusätzlich die Eigenschaft haben, dass es eine ausgezeichnete Ecke gibt, zu der keine Kante führt. Diese Ecke heißt Wurzel und muss eindeutig sein, womit von einem gerichteten Baum gefordert wird, dass keine isolierten Ecken oder Teilgraphen auftreten.

Das Gegenstück zu der Wurzel ist das Blatt. Der Begriff des Blattes ist nicht nur für Bäume, sondern für beliebige ungerichtete und gerichtete Graphen definiert.

- G = (E,K) heißt ein markierter Graph, wenn es Markierungsfunktionen
  - $M_E: E \rightarrow G_E$
  - $M_K: K \rightarrow G_K$

  mit geeigneten Mengen G_E, G_K gibt





- Welche Mengen liegen dem markierten Graph zugrunde:

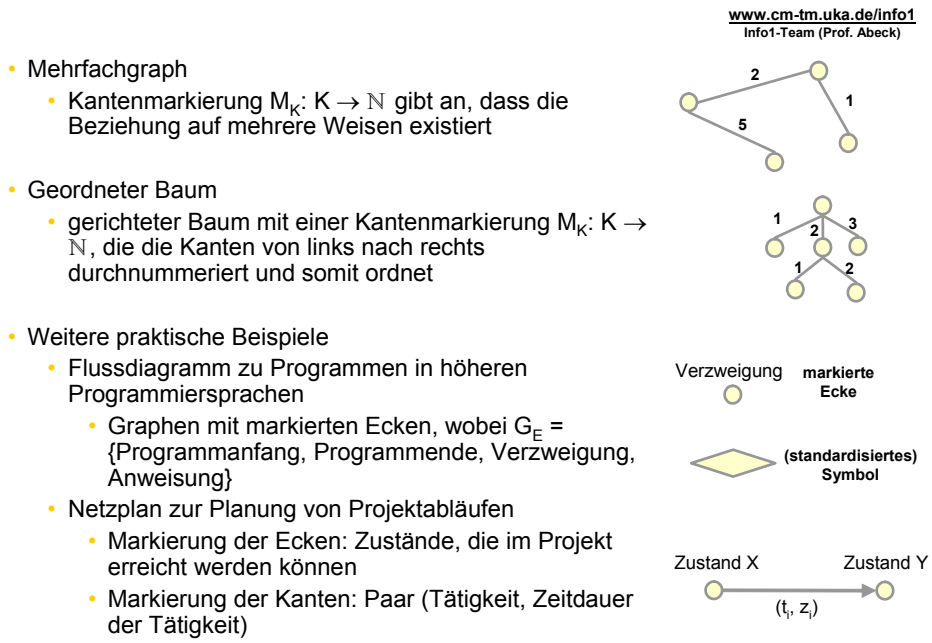
G_E = _____

G_K = _____

Interaktion 15: Markierter Graph

Durch die Zuordnung von Eigenschaften zu Ecken und zu Kanten lässt sich dem Graphen eine bestimmte Bedeutung zuordnen. Diese Zuordnung wird als Markierung bezeichnet, der resultierende Graph heißt entsprechend markierter Graph. Formal bedeutet die Markierung die Einführung von zwei Markierungsfunktionen, einer Eckenmarkierung und einer Kantenmarkierung. Mit der Einführung der Funktionen müssen auch die Wertebereiche festgelegt werden, in die die Ecken bzw. Kanten abgebildet werden sollen.

In Interaktion 15 wird neben der formalen Definition eines markierten Graphen der in Interaktion 9 erstellte Beispielgraph gezeigt, der markiert ist. Die dem Graphen zugrunde liegenden Eckenmarkierungs- und Kantenmarkierungsmengen sind anzugeben.



Information 16: Ausprägungen von markierten Graphen

Die Markierungsfunktionen können für unterschiedliche Zwecke sinnvoll genutzt werden, was zu verschiedenen Ausprägungen von markierten Graphen führt. In vielen Fällen, wie z.B. auch in dem von Euler gelösten Königsberger Brückenproblem, kommt es vor, dass zwei Ecken durch mehr als eine Kante verbunden sind.

Mit einer Kantenmarkierung, die jeder Kante eine natürliche Zahl zuordnet, lassen sich auch so genannte geordnete Bäume erstellen. Beispiele für geordnete Bäume sind Ableitungsbäume oder Kantorowitsch-Bäume.

Von besonderer praktischer Bedeutung sind die Flussdiagramme zu Programmen, die ebenfalls eine bestimmte Ausprägung markierter Graphen darstellen. Die in Information 16 angegebenen Beispiele verdeutlichen, welche umfangreichen Möglichkeiten durch Graphen geboten werden, wenn man den Ecken und/oder Kanten eine bestimmte Semantik zuordnet.
Während in den vorhergehenden zwei Ausprägungen die Kanten markiert wurden, sind es bei den Flussdiagrammen die Ecken, denen jeweils eine von vier Bedeutungen zugewiesen wird, wobei nur eine Ecke mit dem Programmanfang markiert sein darf. Zu den Flussdiagrammen sei Folgendes angemerkt:

(1) Markierung der von der Verzweigungsecke ausgehenden Kanten mit ja (then-Fall) bzw. nein (else-Fall).
(2) Es sind ggf. mehrere Programmende-Ecken möglich (allerdings nur eine Programmanfang-Ecke).
(3) Der Eckentyp legt den Ausgangsgrad eindeutig fest, z.B. $|e^\bullet| = 0$, wenn $M_E(e)$ = Programmende.

Die UML-Diagramme (Aktivitätsdiagramme), die beispielsweise in der Kurseinheit PROGRAMMIERGRUNDLAGEN erstellt werden, sind Beispiele von Flussdiagrammen mit standardisierten graphischen Symbolen.

Ein weiteres und abschließendes Beispiel eines markierten Graphen ist der in zahlreichen Anwendungsszenarien genutzte Netzplan. Ein Netzplan stellt die Ablaufbeschreibung eines Projekts dar. Es handelt es sich hierbei um einen Graphen mit folgenden Eigenschaften:

- Er enthält keine geschlossenen Kantenzüge, d.h. er ist zyklenfrei.
- Er beinhaltet zwei ausgezeichnete Ecken, ähnlich wie das zuvor behandelte Flussdiagramm.
- Diese Ecken sind durch ihren Eingangs- bzw. Ausgangsgrad gekennzeichnet.

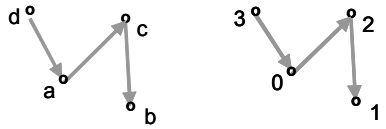
Durch die Markierungsfunktionen wird den Ecken und Kanten folgende Bedeutung zugeordnet:

- Ecken beschreiben die Projektzustände
- Kanten beschreiben, wie man von einem Projektzustand in den nächsten kommt. Hierzu ist eine bestimmte Tätigkeit erforderlich. Zu der Tätigkeit wird zudem die erforderliche Zeitdauer angegeben.

Am Beispiel des Netzplans lässt sich leicht klarmachen, dass durch geeignete Auswertung der im Graph gehaltenen Informationen relevante Fragen zur Projektplanung beantwortet werden können. So ist in einem Projekt zu klären, welche anderen Tätigkeiten vor einer bestimmten Tätigkeit durchgeführt sein müssen und wie lange diese dauern.
Die Bearbeitung von Netzplan-Fragen durch einen Rechner setzt voraus, dass ein Graph geeignet im Speicher dieses Rechner gehalten werden kann, was zu den in Information 17 behandelten Repräsentationsmöglichkeiten von Relationen und Graphen führt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Gegeben sei ein Graph G = (E, K), wobei jede Ecke durchnummeriert ist



- Darstellungsform 1: Adjazenzmatrix
  - n × n - Matrix A = (a_{ij})
  - a_{ij} = 1, wenn (i, j) ∈ K, a_{ij} = 0 sonst
  - anstelle von {0, 1} ist auch ein beliebiger anderer Binärcode möglich

von \ nach	**0**	**1**	**2**	**3**
0	0	0	1	0
1	0	0	0	0
2	0	1	0	0
3	1	0	0	0

- Darstellungsform 2: Adjazenzliste
  - jeder Ecke e_i wird die Menge {e_{i1}, ..., e_{in}} der Ecken e_{ij} mit $e_i \rightarrow e_{ij}$ zugeordnet
  - die jeder Ecke zugeordneten Mengen lassen sich sinnvoll durch Listen wiedergeben

0: [2, 3]
1: []
2: [1]
3: [0]

Information 17: Repräsentation von Relationen und Graphen

Zunächst sind die Bezeichnungen der Ecken geeignet zu nummerieren, d.h. die "erste" Ecke erhält die Nummer 0, die folgende Ecke erhält die Nummer 1 usw.
In den Beispielgraphen würde eine denkbare Durchnummerierung bedeuten, dass der Ecke mit der Bezeichnung a die Nummer 0, der Ecke mit der Bezeichnung b die Nummer 1 usw. zugewiesen wird.

Die Frage ist nun, wie man einen solchen Graphen (bzw. allgemein eine Relation) in einem Rechner repräsentiert. Offensichtlich ist die bislang verwendete graphische Darstellung hierfür nicht geeignet.

Es lassen sich zwei grundsätzliche Formen der Darstellung von Graphen in einem Rechner unterscheiden:

1. Bei der Adjazenzmatrix handelt es sich um eine Matrix mit einer Zeilen- und Spaltenzahl, die der Eckenzahl des Graphen entspricht. Die Zeilen und Spalten der Matrix sind mit den

Nummern der Ecken indiziert. Der Wert eines Adjazenzmatrix-Elements (i, j) bestimmt, ob eine Kante zwischen Ecke i und Ecke j besteht (Wert 1) oder nicht (Wert 0).
Somit ergibt sich für den in Information 17 gezeigten Beispielgraphen die o.a. Adjazenzmatrix. Statt der verwendeten Menge {0, 1} kann auch ein beliebiger anderer Binärcode, wie z.B. {O, L} oder {wahr, falsch} verwendet werden.

2. Die zweite Darstellungsform wird als Adjazenzliste bezeichnet. Bei dieser Darstellungsform eines Graphen werden zu jeder Ecke alle diejenigen Ecken aufgelistet, zu denen eine Kante existiert.
   Das Vorgehen angewendet auf den Beispielgraphen führt zu dem in Information 17 angegebenen Ergebnis.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Es wird davon ausgegangen, dass die Relation als Adjazenzmatrix vorliegt

- Eigenschaften einer Relation
  - Relation reflexiv $\Leftrightarrow a_{ii} = 1$ für alle i = 0, ..., n-1
  - Relation symmetrisch $\Leftrightarrow a_{ij} = a_{ji}$
  - Relation transitiv $\Leftrightarrow$ aus $a_{ij} = 1$ und $a_{jk} = 1$ folgt $a_{ik} = 1$

- Verknüpfung $\rho \circ \sigma$ zweier Relationen $\rho, \sigma \subseteq E \times E$
  - ρ durch Adjazenzmatrix $A= (a_{ij})$, σ durch $B = (b_{ij})$ repräsentiert
  - $C = (c_{ij})$ ist die Adjazenzmatrix der Verknüpfung $\rho \circ \sigma$ mit

$$c_{ij} = \begin{cases} 1, \text{ wenn } \sum_{k=0}^{n-1} a_{ik} b_{kj} \geq 1 \\ 0, \text{ sonst} \end{cases}$$

Information 18: Eigenschaften und Verknüpfung von Relationen

Die Eigenschaften von Relationen spiegeln sich in Form konkreter Werte wider, die die Elemente in der zu der Relation zugehörigen Adjazenzmatrix annehmen:

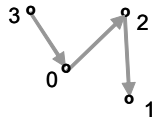
- Eine reflexive Relation ist gegeben, wenn in der Diagonale der Adjazenzmatrix ausschließlich der Wert 1 auftritt.
- Eine symmetrische Relation führt zu einer symmetrischen Matrix, d.h. $a_{ij} = a_{ji}$.
- Die Eigenschaft der Transitivität bedeutet, dass bei einem (aus zwei Kanten bestehenden) Weg von Knoten i zu j und von j zu k gewährleistet ist, dass eine Kante von i zu k besteht.

Information 18 zeigt, wie die Verknüpfung $\rho \circ \sigma$ von zwei über $E \times E$ definierte Relationen ρ und σ festgelegt ist.

Die Eigenschaften der Reflexivität und insbesondere der Transitivität können für eine beliebige Relation bzw. einen beliebigen Graphen erzwungen werden, indem die nachfolgend behandelte reflexive transitive Hülle gebildet wird.

- Eine reflexive transitive Hülle zu einem Graphen entsteht dadurch, dass zu dem Graphen Kanten zwischen jeweils zwei über einen Weg verbundenen Ecken hinzugefügt werden

- Beispiel:



von \ nach	0	1	2	3
0	_	_	1	_
1	_	_	_	_
2	_	1	_	_
3	1	_	_	_

Der Graph und die dazugehörige Adjazenzmatrix sind um alle Kanten zu ergänzen, die zur reflexiven transitive Hülle des Ausgangsgraphen gehören

- Definition der reflexiven transitiven Hülle ρ^* zu einer Relation ρ
  $\rho^* = \{(i,j) \mid$ es gibt einen Weg zwischen Ecke i und Ecke j$\}$

- Bei der transitiven Hülle ρ^+ werden die Schlingen (Wege der Länge 0) nicht hinzugefügt

Interaktion 19: Reflexive transitive Hülle eines Graphen

Wie in Interaktion 19 ausgeführt ist, werden bei der Hüllenbildung Kanten hinzugefügt, um die beiden Eigenschaften zu erfüllen. Am Beispielgraphen, der entsprechend zu ergänzen ist, wird das Vorgehen deutlich gemacht.
Die reflexive transitive Hülle wird durch ein zu der Relationsbezeichnung hinzugefügten Stern, die (nicht-reflexive) transitive Hülle durch ein Additionszeichen kenntlich gemacht.

Von Warshall wurde 1962 ein einfaches, nicht-rekursives Verfahren entwickelt, das die Hüllenberechnung eines Graphen durchführt. Dem Verfahren liegt der folgende offensichtliche Sachverhalt zugrunde [Se02]: Falls ein Weg existiert, um von Ecke i zu Ecke k zu gelangen, und ein Weg, um von k nach j zu gelangen, so existiert auch ein Weg, um von i nach j zu gelangen.

Die Idee, die zum Warshall-Algorithmus führt, besteht darin, die Weglänge und die darin benutzten Eckennummern schrittweise (jeweils um 1) zu erhöhen. Aus dieser Überlegung resultiert die in Information 20 angegebene Relation σ.

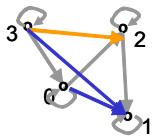
- Verfahren gemäß Warshall

  Gegeben sei eine reflexive Relation ρ über einer endlichen Eckenmenge E = {0, ..., n-1} mit Adjazenzmatrix A

  $\sigma^{(k)}$ bezeichne die Relation

  $\sigma^{(k)} = \{(i,j) \mid$ es gibt einen Weg $i \rightarrow e_1 \rightarrow ... \rightarrow e_{l-1} \rightarrow j$, $l \leq k+2$ und $e_r \in \{0, ..., k\}$ für $1 \leq r \leq l-1\}$

- Beispiel:



$\sigma^{(0)}$

1	0	1	0
0	1	0	0
0	1	1	0
1	0	1	1

$\sigma^{(1)}$

1	0	1	0
0	1	0	0
0	1	1	0
1	0	1	1

$\sigma^{(2)}$

1	1	1	0
0	1	0	0
0	1	1	0
1	1	1	1

$\sigma^{(3)}$

1	1	1	0
0	1	0	0
0	1	1	0
1	1	1	1

Information 20: Berechnung der reflexiven transitiven Hülle

Durch den Parameter k werden alle Wege, die in dem durch die Relation ρ vorgegebenen Graphen bestehen, sukzessive gefunden. Wie die Definition der zu diesem Zweck eingeführten Relation σ verdeutlicht, wird die Suche durch k in den folgenden zwei Punkten eingeschränkt:

1. Durch $l \leq k+2$ wird erreicht, dass der Weg von einem Anfangs- zu einem Endeknoten nur aus k Zwischenknoten bestehen darf (Weglängenbegrenzung).
2. Die Festlegung $e_r \in \{0, ..., k\}$ besagt, dass als Zwischenknoten nur die Knoten zwischen 0 und höchstens k auftreten dürfen (Knotennummernbegrenzung).

Anhand des einfachen Beispielgraphen wird in Information 20 der schrittweise Aufbau der Relation entlang des Wertes k dargestellt.

- Anforderungsbeschreibung
  - Eingabe: Adjazenzmatrix A einer Relation σ
  - Ausgabe: Adjazenzmatrix S von σ^*

- Algorithmusbeschreibung in Pseudocode

```
S := A
für i = 0, ..., n-1  setze s_ii := 1
für k = 0, ..., n-1
  für i = 0, ..., n-1
    für j = 0, ..., n-1
      falls (s_ii  + s_ik * s_ki) >= 1 setze s_ii := 1
```

Information 21: Warshall-Algorithmus

Die Umsetzung dieses Berechnungsvorgehens auf entsprechende auf der Adjazenzmatrix wirkende elementare Operationen zeigt Information 21. Der Algorithmus ist zwar einfach in der Aufschreibung, aber aufgrund der drei geschachtelten Schleifen komplex im Hinblick auf dessen Berechnungsdauer.

Der in Pseudocode angegebene Warshall-Algorithmus lässt sich mittels der Sprachelemente, die in den Kurseinheiten PROGRAMMIERGRUNDLAGEN und IMPERATIVE PROGRAMMIERUNG eingeführt werden, in ein Java-Programm umsetzen.

2 ALGEBREN

In diesem Kapitel werden die bislang eingeführten Strukturen verallgemeinert, um dadurch die mathematische Grundlage für die Beschreibung der in der Informatik relevanten Systeme zu legen. Hieraus resultieren die Algebren, zu denen Information 22 einen Überblick liefert [Go97].

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- In der Informatik werden Algebren zur Beschreibung von formalen Systemen und Datenstrukturen genutzt
  - durch Ausführung einer Operation wird ein System von einem Zustand in einen Folgezustand überführt
- Unterschied zwischen den in der Mathematik und in der Informatik betrachteten Algebren
  - die Operationen in Informatiksystemen sind oft wesentlich komplizierter als die Operationen in Gruppen, Ringen, Körpern , ...
- Es ist zwischen dem Aufbau und dem berechneten Ergebnis einer algebraischen Formel zu unterscheiden
  - der Algorithmus beschreibt den Lösungsweg
- Eine für die Informatik wichtige Algebra ist die Boolesche Algebra

Information 22: ALGEBREN - Überblick

Ein Beispiel eines zu beschreibenden Informatiksystems ist das in der Kurseinheit EINFÜHRUNG IN DAS FACHGEBIET beschriebene Ablagesystem. Operationen hierauf sind beispielsweise das Anmelden oder das Abrufen von Inhalten. Das Beispiel zeigt bereits, dass die in der Informatik betrachteten Operationen im Vergleich zu den aus der Mathematik bekannten Operationen vielfältiger und höherwertiger sind. Die höherwertigen Operationen eines Informatiksystems werden durch entsprechende Sprachtransformationen (Übersetzung) in einfachere Operationen bis auf die Ebene der Maschinenoperationen umgesetzt.

Die Formel, die im Allgemeinen als Term bezeichnet wird, ist in einer Algebra ein zentraler Begriff.

Die Menge der aus einer Algebra hervorgehenden Terme bildet ebenfalls eine Algebra, die so genannte Termalgebra. Die Termalgebra ist für die Systemmodellierung und die praktische Realisierung wichtiger als die Algebra, aus der sie hervorgeht.

Zwischen diesen beiden Ebenen liegt eine weitere in der Informatik zu berücksichtigende Ebene der Realisierung. Ein Algorithmus liefert neben dem Ergebnis auch den Lösungsweg, der zum Ergebnis führt. Dabei ist dieser Weg nicht zwingend eindeutig, da Gesetze (wie z.B. das Distributivgesetz) alternative Vorgehensweisen ermöglichen. Außerdem führen auch unterschiedliche Repräsentationen (z.B. Graph als Adjazenzmatrix oder Adjazenzliste) zu weiteren Alternativen der Realisierung. Daraus ergibt sich die Notwendigkeit, zwischen der Spezifikation eines Algorithmus und dessen tatsächlicher Realisierung sauber zu trennen.

Für die Informatik hat die Boolesche Algebra, auf die in diesem Kapitel näher eingegangen wird, eine besondere Bedeutung.

2.1 Formeln

Formeln werden gebildet auf der Grundlage von Operationen und der Signatur einer Algebra (siehe Information 23).

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Abbildung $f:A^n \rightarrow A \quad f(a_1,....,a_n)$
  - n-stellige Operation auf A
  - Operanden a_i heißen Argumente der Operation
  - n heißt Stelligkeit von f
    - n=1: unäre Operation
    - n=2: binäre Operation

- Beispiel: Potenzmenge $A = \mathcal{P}(U)$
  - U: Grundmenge
  - Operationen im booleschen Verband über $\mathcal{P}(U)$:
    - unär: $h(M) = \complement M$
    - binär: $f(M,N) = M \cup N \quad g(M,N) = M \cap N$

- Signatur $\Sigma = \Sigma^{(0)} \cup \Sigma^{(1)} \cup \Sigma^{(2)} \cup .....$
  - $f \in \Sigma^{(n)}$ wenn n die Stelligkeit von f ist

Information 23: Operationen, Signatur

Um eine Abbildung definieren zu können, wird eine Menge A von Elementen benötigt. Aus dieser Menge werden n Elemente herausgegriffen und auf ein Element dieser Menge abgebildet.

Die Elemente, auf der die Abbildung arbeitet, heißen Argumente. Die Anzahl der Elemente heißt die Stelligkeit der Abbildung. Einstellige bzw. zweistellige Abbildungen werden als unäre bzw. binäre Operationen bezeichnet.

Beispiele für solche unären bzw. binären Operationen liefert der boolesche Verband über der Potenzmenge $\mathcal{P}(U)$ der Grundmenge U. Die Potenzmenge $\mathcal{P}(U)$ beinhaltet sämtliche Teilmengen aus der Grundmenge U (inklusive der leeren Menge). Hier ist also jeweils eine Teilmenge ein Argument der Abbildungen. Beispiel einer einstelligen, also unären Operation ist die Komplementbildung. Beispiele für zweistellige Operationen im booleschen Verband sind die Vereinigung und der Durchschnitt zweier Mengen. Diese Operationen lassen sich auf n-stellige Operationen ausdehnen.

Mit der Signatur wird eine Notation zur Beschreibung solcher Strukturen, die aus Mengen und darauf definierten Abbildungen bzw. Operationen entstehen, eingeführt. Die Signatur Σ ist eine nach Stelligkeit geordnete Auflistung der Operationen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Boolesche Algebra
  - $\mathcal{B} = \mathcal{B}(A, \bot, \top, \complement, \vee, \wedge)$
- Zugehörige Signatur
  - $\sum = \{\bot, \top, \complement, \vee, \wedge\}$
  - $\sum^{(0)} = \{\bot, \top\}$ $\sum^{(1)} = \{\complement\}$ $\sum^{(2)} = \{\vee, \wedge\}$
- $f^{(n)}$ oder f/n ist eine Kurzschreibweise von $f \in \sum^{(n)}$
- Operationen aus $\sum^{(0)}$
  - haben keine Argumente und liefern daher immer dasselbe Resultat op = c
    - diese Operationen heißen daher Konstante
    - werden als Operanden aufgefasst
  - es muss mindestens eine solche Operation vorhanden sein

Information 24: Beispiel einer Signatur

Ein wichtiges Beispiel einer Algebra, zu der in Information 24 die Signatur angegeben wird, ist die Boolesche Algebra. Dabei bezeichnet A die Menge der Bezeichnungen, die in den booleschen Ausdrücken auftreten dürfen.

Die danach folgenden Operationen lassen sich gemäß ihrer Stelligkeit in disjunkte Operationenmengen aufteilen. Die Notation $\Sigma^{(i)}$ liefert eine disjunkte Zerlegung der Operationen der Signatur, wobei i die Stelligkeit der auftretenden angibt.

Im ersten Moment erscheint eine nicht endliche Anzahl von Operationen nicht sinnvoll; dieser Sachverhalt zielt auf die konstanten Operationen $\Sigma^{(0)}$ ab, die weiter unten näher betrachtet werden.

Im Beispiel sind die 0-stelligen Operationen der booleschen Algebra die Konstanten $\bot$ ("Bottom") und $\top$ ("Top"), als einzige 1-stellige Operation tritt $\complement$ ("Komplement") auf und die zwei zweistelligen Operationen sind $\wedge$ ("und") und $\vee$ ("oder").
Für $f \in \Sigma^{(n)}$ kann eine vereinfachende Kurzschreibweise verwendet werden, wie in Information 24 ausgeführt ist.

Die 0-stelligen "Operationen", also $\Sigma^{(0)}$, haben keine Argumente, weshalb sie immer dasselbe Ergebnis liefern. Sie werden als Konstanten bezeichnet und sind innerhalb einer Formel bzw. eines Terms elementare, nicht weiter zerlegbare Operanden. Warum die Menge der Konstanten nicht leer sein darf, wird anhand der in Interaktion 25 vorgestellten Bildungsgesetze für Terme deutlich.

- $(M \wedge \top) \vee \complement P$
  - ist ein Beispiel einer Formel zur Signatur $\sum = (\bot, \top, \complement, \vee, \wedge)$
  - heißt auch Term oder Ausdruck
  - Operationen: _____ Operanden: _____

- Ein (korrekter) Term ist
  (1) entweder eine Konstante $a \in X$
  (2) oder die Anwendung eines Operators f(a,b,...) mit f, wobei die Anzahl der Operanden a,b,... der Stelligkeit von f entspricht und jeder Operand wieder ein korrekter Term ist

- Unterterm
  - Term, der als Argument eines anderen Terms auftritt

- n-stelliger Term
  - Term, der durch Anwendung einer n-stelligen Operation entsteht

Interaktion 25: Formel und Term

Auf der Basis der Signatur wird der Begriff der Formel eingeführt. Formeln bestehen aus nicht weiter zerlegbaren Operanden, die durch die in der Signatur angegebenen Operationen unter Berücksichtigung von deren Stelligkeit verknüpft sind. Bei der in Interaktion 25 angegebenen Beispielformel ist zu beachten, dass das Element $\top$ nicht als (0-stellige) Operation, sondern als elementarer Operand aufgefasst wird.

Für den Begriff der Formel existieren noch weitere Begriffe wie Term und Ausdruck, die zunächst alle synonym verwendet werden. In der Beispielformel hat der elementare, nicht weiter zerlegbare Operand $\top$ die Bedeutung, dass er das größte Element bezeichnet.

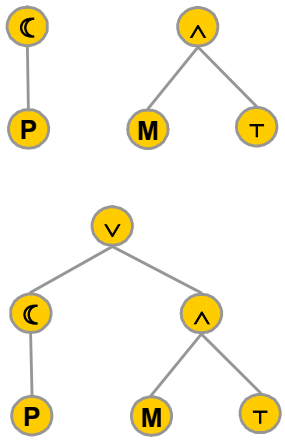
Es bestehen zwei in Interaktion 25 angegebene Regeln, gemäß derer ein korrekter Term (bzw. eine korrekte Formel) gebildet werden kann. Im Englischen werden solche Ausdrücke als *well-formed formula* (bzw. *well-formed term*) bezeichnet.

Diese als Operanden enthaltenen Terme werden auch als Unterterme bezeichnet. Ein Term, der auf n solchen Untertermen und einer n-stelligen Operation aufgebaut ist, wird entsprechend als n-stelliger Term bezeichnet.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- n-stellige Operation f wird als Baum der Höhe 1 dargestellt
  - Wurzel wird mit dem Operationssymbol bezeichnet
  - wird als Kantorowitsch-Baum bezeichnet
- Zusammenfügen der einzelnen Operations-Bäume

Warum gilt die Baum-Eigenschaft?



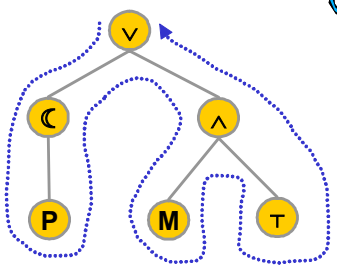
Interaktion 26: Termdarstellung als Kantorowitsch-Baum

Terme lassen sich auch graphisch in Form von so genannten Kantorowitsch-Bäumen darstellen. Der Aufbau dieser Bäume erfolgt entlang des Termaufbaus beginnend bei den Konstanten, die einzelne Ecken darstellen und den Operationen, die als Kantorowitsch-Bäume der Höhe 1 dargestellt werden.

Zur Darstellung komplexerer Terme, wie der zuvor eingeführten Beispiel-Term $(M \wedge \top) \vee \complement P$ sind Kantorowitsch-Bäume geeignet zusammen zu setzen. Gemäß den Ausführungen im Kapitel zu den Relationen (siehe Interaktion 13) erfüllen die Kantorowitsch-Bäume die an Bäume gestellten Eigenschaften, wie in Interaktion 26 näher ausgeführt werden soll.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Infix a f b
  - nur für binäre Operationen
  - z.B. a+b, M∩N
  - beim Durchlauf durch den Kantorowitsch-Baum werden die Ecken bei deren vorletzten Besuch angeschrieben (bzw. beim ersten Besuch, falls nur einmal besucht)
- Präfix f a b
  - Operationssymbol vorne
  - z.B. + 2 3, ∁N
  - Anschreiben beim ersten Besuch
- Postfix a b f
  - Operationssymbol hinten
  - z.B. A B union
  - Anschreiben beim letzten Besuch



Die gepunktete Linie beschreibt den Durchlauf durch der Kantorwitsch-Baum

Infix: _____

Präfix: _____

Postfix: _____

Interaktion 27: Schreibweisen für Operationen

Der Kantorowitsch-Baum ist auch dazu nützlich, die verschiedenen in Interaktion 27 aufgezeigten Schreibweisen für Operationen zu verdeutlichen. Jede der drei genannten wichtigsten Schreibweisen Infix, Präfix und Postfix entspricht einer Strategie, gemäß der die Ecken beim Durchlauf durch den Kantorowitsch-Baum aufgeschrieben werden.

- Zwei weitere Schreibweisen
  - Funktionsform
    - ähnlich der Präfix-Notation
    - z.B. union(X, Y)
  - Spezialnotation
    - nur für unäre Operationen
    - z.B. $\bar{E}$ (Komplement)

- Lösungen zur Aufhebung von Mehrdeutigkeiten
  - Vorrangregeln zwischen Operatoren
  - Klammerung

Information 28: Weitere Schreibweisen und Vorrangregeln

Daneben bestehen mit der Funktionsform und den Spezialnotationen noch zwei weitere Schreibweisen (siehe Information 28). Die Funktionsform weist bzgl. der Stellung des Operationssymbols Ähnlichkeiten mit der Präfix-Form auf. Die Spezialnotation (wozu beispielsweise auch die Wurzelschreibweise gehört) tritt nur bei unären Operatoren auf.

Ein Beispiel für eine Vorrangregel ist, dass * stärker bindet als +. Zur Auflösung von Mehrdeutigkeiten durch Klammern werden in der Informatik ausschließlich die runden Klammern benutzt, da anderen Klammerformen (z.B. geschweift oder eckig) für andere Zwecke genutzt werden.

2.2 Boolesche Algebra

Die boolesche Algebra ist ein vollständiger, komplementärer, distributiver Verband. In Information 29 sind alle Gesetze, die in dieser Algebra mit der angegebenen Signatur gelten, aufgeführt.

- Signatur der booleschen Algebra $\mathcal{B} = \mathcal{B}(A,\bot,\top,\complement,\vee,\wedge)$
  - $\bot$ ist kleinstes Element und $\top$ ist größtes Element

- Gesetze $(x, y, z \in A)$

V1 Assoziativität	$(x\wedge y)\wedge z = x\wedge(y\wedge z)$	$(x\vee y)\vee z = x\vee (y\vee z)$
V2 Kommutativität	$x\wedge y = y\wedge x$	$x\vee y = y\vee x$
V3 Idempotenz	$x\wedge x = x$	$x\vee x = x$
V4 Verschmelzung	$(x\vee y)\wedge x = x$	$(x\wedge y)\vee y = y$
V5 Distributivität	$x\vee (y\wedge z) = (x\vee y)\wedge(x\vee z)$	$x\wedge(y\vee z) = (x\wedge y)\vee(x\wedge z)$
V6 Modularität	falls $z \leq x$ gilt:	
	$x\wedge(y\vee z) = (x\wedge y)\vee z$	
V7 Neutrales Element	$x\wedge\bot = \bot$	$x\vee\bot = x$
	$x\wedge\top = x$	$x\vee\top = \top$
V8 Komplement	$x\wedge\complement x = \bot$	$x\vee\complement x = \top$
V9 Involution	$\complement(\complement x) = x$	
V10 DeMorgan	$\complement(x\wedge y) = \complement x\vee\complement y$	$\complement(x\vee y) = \complement x\wedge\complement y$

Information 29: Signatur und Gesetze der booleschen Algebra

Wegen V9 ist jedes Element $x\in A$ ein Komplement eines anderen Elements $x' = \complement x$. Hieraus resultiert das in der booleschen Algebra geltende Dualitätsprinzip.

Durch die zehn aufgeführten Vorschriften ist die boolesche Algebra überspezifiziert, d.h. man kann auch gewisse Gesetze weglassen, da sich diese zwangsläufig aus den übrigen Gesetzen ableiten lassen. So kann man beispielsweise zu den Vorschriften V1 bis V5 entweder jeweils das erste oder jeweils das zweite Gesetz ersatzlos streichen.

- Die Mengenalgebra ist ein bekanntes Beispiel einer booleschen Algebra
  - Gegeben: eine beliebige Menge U
  - $(\mathcal{P}(U),\cup,\cap,\setminus)$ ist eine boolesche Algebra und heißt Mengenalgebra, wenn $\cup$ die Mengenvereinigung, $\cap$ der Mengendurchschnitt und $\setminus$ das Mengenkomplement ist

- $\mathcal{L}$ ist ein Mengensystem, das als eine Teilmenge (von Mengen) aus der Potenzmenge $\mathcal{P}(U)$ hervorgeht
  - $\mathcal{L}$ ist eine Unteralgebra, falls diese die an eine Algebra gestellte Anforderung erfüllt
  - Welche Anforderung muss $\mathcal{L}$ erfüllen?

  _____

- Bei endlicher Grundmenge A sind die Begriffe boolesche Algebra und Mengenalgebra äquivalent
  - formuliert im Satz von Stone

Interaktion 30: Mengenalgebra und Unteralgebra $\mathcal{L}$

Das bekannteste Beispiel einer booleschen Algebra ist die Mengenalgebra, die auf der Potenzmenge einer beliebigen endlichen Menge definiert ist und der die bekannten Mengenoperationen (Vereinigung, Durchschnitt, Komplement) zugrunde gelegt sind.

Eine Unteralgebra $\mathfrak{L}$ wird dadurch gebildet, dass aus der Potenzmenge (Menge aller Teilmengen) nur gewisse dieser Teilmengen herausgegriffen werden. Diese Teilmenge muss die an eine Algebra gestellte Anforderung erfüllen, nach der in Interaktion 30 gefragt wird.

Im Satz von Stone ist der Übergang von der booleschen Algebra zur Mengenalgebra und umgekehrt durch bijektive Abbildungen festgelegt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Betrachtet wird die Mengenalgebra (boolesche Algebra) über einer einelementigen Grundmenge U = {a}
  - $\mathcal{P}(U) = \{\varnothing,\{a\}\}$ ist zweielementig
- Diese boolesche Algebra wird mit $\mathcal{B} = \{0, L\}$ bezeichnet
  - 0 heißt Nullelement, L heißt Einselement
- $\mathcal{B}$ bildet die Grundlage der Aussagenlogik und aller digitalen Codierungen und Schaltungen
- Kartesisches Produkt $\mathcal{B}^n = \{0,L\}^n = \mathcal{B} \times ... \times \mathcal{B}$
  - Elementweises Ausführen der Operationen auf den n-Tupeln $(a_1,...,a_n)$, $a_i \in \{0,L\}$ ergibt, dass auch $\mathcal{B}^n$ eine boolesche Algebra ist

Information 31: Spezielle boolesche Algebra

Die Potenzmenge einer 1-elementigen Grundmenge besteht aus
(1) dem Nullelement, also der leeren Menge $\varnothing$,
(2) dem Einselement {a}, also der gesamten Grundmenge.

Aufgrund dieser speziellen Eigenschaft und der großen Bedeutung, die gerade diese Ausprägung einer booleschen Algebra in der Informatik hat, wird hierfür eine spezielle Bezeichnung $\mathcal{B}$ eingeführt.

Konkret bildet diese zweiwertige boolesche Algebra die Grundlage für die in der Kurseinheit RECHENSTRUKTUREN UND FUNKTIONALE PROGRAMME behandelte Aussagenlogik sowie für alle digitalen Codierungen und Schaltungen.

Die Operation des Kartesischen Produkts führt bei Anwendung auf die zweiwertige boolesche Algebra $\mathcal{B}$ zu n-Tupeln, wobei diese Konstruktion $\mathcal{B}^n$ wieder eine boolesche Algebra ergibt.

Abschließend wird in Interaktion 32 eine wichtige Normalform zu Termen der booleschen Algebra vorgestellt.

- Ziel ist die standardisierte Darstellung von booleschen Termen

- Disjunktive Normalform
  - durch Disjunktionen ($\vee$) verknüpfte Teilterme
  - Teilterme bestehen aus durch Konjunktionen ($\wedge$) verknüpften negierten oder nicht-negierten Elementen
  - Beispiel: $(a \wedge b \wedge c) \vee (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c)$

- Konjunktive Normalform
  - Konjunktion von Teiltermen, die aus disjunktiv verknüpften negierten oder nicht-negierten Elementen zusammengesetzt sind
  - Beispiel: $(a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c) = f(a, b, c)$

Wertetabelle:

a	O	O	O	O	L	L	L	L
b	O	O	L	L	O	O	L	L
c	O	L	O	L	O	L	O	L
f								

Interaktion 32: Disjunktive und konjunktive Normalform

Die disjunktive Normalform erhält ihren Namen aufgrund der durch Disjunktionen (also durch die $\vee$-Operationen) verbundenen Teiltermen. Die von der Normalform zu erfüllende Anforderung steckt dabei in der scharfen Restriktion der disjunktiv verknüpften Teilterme. Es wird nämlich von jedem Term verlangt, dass

- ausschließlich die Konjunktion (also die $\wedge$-Operation) sowie die Negation auftreten darf und
- keine weiteren geklammerten Teilterme enthalten sein dürfen, d.h. der Teilterm ausschließlich aus Elementen aufgebaut sein darf.

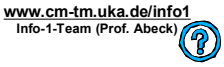
Die konjunktive Normalform ist die duale Form der disjunktiven Normalform, d.h. es handelt sich um eine Konjunktion von Teiltermen, die aus disjunktiv verknüpften negierten oder nicht-negierten Elementen zusammengesetzt sind.

Die Beispiele verdeutlichen den Aufbau einer disjunktiven bzw. konjunktiven Normalform. Anhand der auszufüllenden Wertetabelle kann man sich den Zusammenhang zwischen den Normalformen und dem Werteverlauf von "normalisierten" booleschen Termen klar machen.

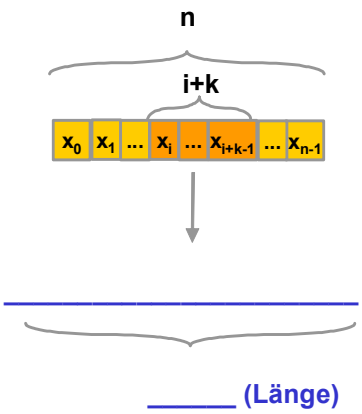
Algebren entsprechen in der Informatik den Rechenstrukturen. Dieser Zusammenhang wird in der Kurseinheit RECHENSTRUKTUREN UND FUNKTIONALE PROGRAMME aufgegriffen und am Beispiel verschiedener für die Informatik besonders wichtigen Algebren verdeutlicht.

3 SEMI-THUE-SYSTEME

Systeme zur Textersetzung, so genannte Semi-Thue-Systeme wurden bereits 1914 von Axel Thue (1863 - 1922, norwegischen Mathematiker und Logiker) entwickelt [Go97].



- Textersetzung mittels Semi-Thue-Systemen stellt die einfachste Form von Algorithmen dar
- Vorgehen
  - endlicher Zeichenvorrat Σ
  - Wörter $x = x_0....x_{n-1}$, wobei Zeichen $x_i \in \Sigma$
  - $|x| = n$ heißt Länge eines Wortes
  - Überführung eines Wortes x in ein anderes Wort durch Ersetzen
    - von Teilwörter $x_i...x_{i+k-1}$ durch andere Wörter y_j .... y_{j+l-1}, wobei $k, l \geq 0$ und $i+k \leq n$



- Schreibweise: $l_1...l_n \rightarrow r_1...r_m$
  - $l_1...l_n$ heißt linke Seite l
  - $r_1...r_m$ heißt rechte Seite r

Interaktion 33: SEMI-THUE-SYSTEME – Prinzip

Semi-Thue-Systeme zeichnen sich nicht nur durch ihre Einfachheit, sondern auch durch ihre hohe Allgemeinheit aus. Es wird von einem endlichen Zeichenvorrat und einer abzählbaren (ggf. unendlichen) Anzahl an Regeln ausgegangen. Die einzige (einfache und allgemeine) Operation, die zugelassen wird, besteht in der Ersetzung von Teilworten: Befindet sich in einem gegebenen Wort x ein Teilwort (hier der Länge k), das die linke Seite einer Regel ist, so kann dieses Teilwort gegen die rechte Seite in x ersetzt werden. Eine solche einfache Textersetzung soll in Interaktion 33 exemplarisch durchgeführt werden und die Länge des sich ergebenden Textes ist anzugeben.

3.1 Regeln und Metaregeln

In Information 34 wird ein einfaches Semi-Thue-System beschrieben, das eine Addition auf der Basis einer Strichdarstellung durchführt. Als Zeichenvorrat werden nur zwei Zeichen, der Strich und das Additionszeichen, benötigt.

- Zeichenvorrat $\Sigma = \{|,+\}$
- Regeln
  (1) $+| \rightarrow |+$
  (2) $+ \rightarrow \varepsilon$
- Bedeutung der Regeln, also des Algorithmus
  - Addition natürlicher Zahlen, die als Folge von Strichen dargestellt sind
  - Beispiel zur Anwendung der Regeln

    $|||+|| \Rightarrow ||||+| \Rightarrow |||||+ \Rightarrow |||||$
- $l \Rightarrow r$ heißt Transformation oder direkte Ableitung
  - $l \Rightarrow^+ r$ heißt: r kann aus l durch fortgesetzte Ableitung gewonnen werden
  - $l \Rightarrow^* r$ heißt: $l \Rightarrow^+ r$ oder $l = r$ (r kann auf l reduziert werden)

Information 34: Beispiel eines Semi-Thue-Systems

Das Ergebnis der Addition von zwei Strichzahlen, die durch ein Additionszeichen getrennt sind, wird durch Wegstreichen des Additionszeichens und Zusammenschieben der beiden Strichdarstellungen erzielt.

Wurde der Übergang von l nach r durch die Anwendung einer einzigen Regel erreicht (z.B. $l = |||+||$ und $r = ||||+|$), so wird von einer direkten Ableitung gesprochen und der Doppelpfeil $\Rightarrow$ benutzt. Wurden eine oder mehrere Regeln angewendet, so ist das eine so genannte fortgesetzte Ableitung, was durch eine Ergänzung des Doppelpfeils um ein hochgestelltes Additionszeichen, also $\Rightarrow^+$, kenntlich gemacht wird. Falls auch der Fall eingeschlossen sein soll, dass keine Regel angewendet werden kann, d.h. $l = r$ ist, so wird das Additionszeichen durch einen Stern ergänzt ($\Rightarrow^*$).

In dem in Information 34 beschriebenen Ableitungsbeispiel wurde die Regel (1) solange angewendet, bis die linke Seite dieser Regel nicht mehr im Text auftrat und damit die Regel auch nicht mehr genutzt werden konnte. Wie durch die in Interaktion 35 angegebenen Metaregeln festgelegt wird, hätte man auch mit der Anwendung der Regel (2) beginnen können (was zu dem gleichen Endresultat führt).

www.cm-tm.uka.de/info1
Info-1-Team (Prof. Abeck)

- Durch folgende drei Metaregeln wird die Anwendung der Regeln festgelegt:

  (M1) wenn a...b → c...d anwendbar ist, ersetze das Teilwort a...b von x durch c...d

  (M2) wenn a...b mehrfach vorkommt oder mehrere Regeln anwendbar sind, so wähle das Teilwort bzw. die Regel beliebig

  (M3) wiederhole die Anwendung von Regeln beliebig oft

Zu dem angegebenen Wort III+II+I ist zu zeigen, dass im Beispiel-Semi-Thue-System beide Formen des Nichtdeterminismus aus Metaregel (M2) vorkommen

III+II+I ⇒ _____

oder ⇒ _____

oder ⇒ _____

oder ⇒ _____

oder ⇒ _____

Interaktion 35: Metaregeln eines Semi-Thue-Systems

Die erste Metaregel (M1) legt das grundsätzliche Prinzip der Textersetzung fest, das am Anfang dieses Abschnitts vorgestellt wurde.
Die zweite Metaregel (M2) betrifft Fälle, in denen eine oder mehrere Regeln an mehreren Stellen zur Anwendung kommen können. Aus diesem Grund weisen Semi-Thue-Systeme ein nichtdeterministisches Verhalten auf.
Die dritte Meta-Regel (M3) realisiert eine Schleife, durch die eine fortlaufende Regel-Anwendung gewährleistet ist, bis keine passende Regel mehr gefunden werden kann.

In der in Information 34 beschriebenen Beispiel-Ableitung entstand ein Nichtdeterminismus dadurch, dass Regel (1) und Regel (2) beide angewendet werden konnten. In der in Interaktion 35 zu ergänzenden Ableitung ergibt sich der Nichtdeterminismus nicht nur aufgrund der Anwendungsmöglichkeiten mehrerer Regeln, sondern zudem aufgrund mehrerer möglicher Anwendungsstellen einer Regel.

Neben der Eigenschaft des Nichtdeterminismus, der eine Aussage zum Verhalten eines Algorithmus trifft, ist die Eigenschaft der Nichtdeterminiertheit zu unterscheiden, die das Ergebnis eines Algorithmus zu einer Eingabe betrifft [Sa04].

3.2 Zusammenhang zu Sprache und Algorithmus

Ein Semi-Thue-System ist somit definiert durch die dem System zugrunde liegende Regelmenge und die für alle Systeme geltenden Metaregeln. Anstelle der Bezeichnung Semi-Thue-System ist auch die Bezeichnung Textersetzungssystem (*String Replacement System*, *String Rewrite System*) üblich.

www.cm-tm.uka.de/info1
Info-1-Team (Prof. Abeck)

- Eine Menge T = {p → q} von Regeln zusammen mit den Metaregeln M1, M2 und M3 heißt ein Semi-Thue-System

Beispiel-Semi-Thue-System T_B

T_B = { _____ }

- Die Menge aller Texte r, die aus dem Text I abgeleitet werden können, heißt die Formale Sprache
  - Schreibweise $L_I = L(T, I)$

$L(T_B, |||+||) =$

{ _____ }

- $T^{-1} = \{q \rightarrow p\}$
  - inverses Semi-Thue-System, das dadurch entsteht, dass alle Pfeilrichtungen umgekehrt werden

T_B^{-1} = { _____ }

$L(T_B^{-1}, |||||) =$

{ _____ }

Interaktion 36: Semi-Thue-System und Formale Sprache

Ein Semi-Thue-System ist ein Beispiel eines Formalen Systems. Unmittelbar verknüpft mit den Formalen Systemen sind die Formalen Sprachen, wie anhand des Beispiel-Semi-Thue-Systems in Interaktion 36 verdeutlicht wird. Das Semi-Thue-System erzeugt aus einem vorgegebenen Text weitere Texte. Diese Textmenge ist die durch das Semi-Thue-System erzeugte formale Sprache.

Es lässt sich zu jedem Semi-Thue-System T sein Inverses T^{-1} bilden, indem einfach die linken und rechten Seiten der Regeln vertauscht werden. T^{-1} ist dann selbst wieder ein Semi-Thue-System.

www.cm-tm.uka.de/info1
Info-1-Team (Prof. Abeck)

- Ein Semi-Thue-System stellt die Grundform eines Algorithmus dar
  - Ersetzungsregeln sind die Operationen, die im Prinzip beliebig oft auf eine Eingabe angewendet werden
  - Terminierung des Algorithmus, sobald keine Ersetzungsregel mehr anwendbar ist
  - Semi-Thue-Systeme sind (potenziell) nicht-terminierende Algorithmen
- Semi-Thue-Systeme sind im Allgemeinen nicht-deterministische Algorithmen
  - siehe Meta-Regel (M2)
  - ein nicht-deterministischer Algorithmus kann in Abhängigkeit der (nicht-deterministisch) gewählten Operationen terminieren oder nicht terminieren

Information 37: Semi-Thue-System und Algorithmus

Semi-Thue-Systeme stellen die einfachste Form eines Algorithmus dar. Die Regeln entsprechen hierbei den Operationen und die Meta-Regeln legen die Ablaufstruktur fest.

Ein Algorithmus terminiert nicht, wenn er für eine Eingabe eine unendliche Folge von Operationen erzeugt. Eine solche Folge kann zustande kommen, da die Ablaufstruktur eine im Prinzip beliebige, ggf. auch unendliche Anzahl von Ausführungen derselben Operation ermöglicht.

Semi-Thue-Systeme können so konstruiert werden, dass für eine gewisse Eingabe nie eine Ableitungssituation erreicht wird, die keine weitere Regelanwendung mehr zulässt. In diesem Fall stellt das Semi-Thue-System mit dieser Eingabe einen nichtterminierenden Algorithmus dar. Er liefert für diese Eingabe dann ein undefiniertes Ergebnis.

Der Sachverhalt der Nichtterminierung wird noch dadurch verschärft, dass Semi-Thue-Systeme auch aufgrund der zweiten Meta-Regel nichtdeterministisch sind. Dadurch kann es möglich sein, dass für ein und dieselbe Eingabe eine Terminierung erfolgt oder nicht erfolgt. In diesem Falle liefert das System ein nichteindeutiges Ergebnis.

3.3 Beispiel Kaffeedosenspiel

In Interaktion 38 wird das durch Regeln mechanisch ausführbare so genannte Kaffeedosenspiel in Form eines Semi-Thue-Systems formuliert. Gegenstand des Spiels ist eine Dose mit schwarzen und weißen Bohnen. Außerdem stehen zusätzliche schwarze Bohnen in ausreichender Anzahl zur Verfügung.

Die Bohnen lassen sich durch einen 2-elementigen Zeichenvorrat Σ beschreiben. Die Umsetzung der zwei Spielregeln in insgesamt vier Semi-Thue-Regeln erfolgt dann auf der Grundlage dieses Zeichenvorrats.

www.cm-tm.uka.de/info1
Info-1-Team (Prof. Abeck)

- Gegeben: Eine Dose mit beliebig angeordneten weißen und schwarzen Bohnen

Formulierung als Semi-Thue-System

Zeichenvorrat Σ = _____
(Eingabe ist der initiale Doseninhalt)

- Spielregeln
  Es sind fortgesetzt blind zwei Bohnen aus der Dose zu nehmen
  (1) Falls die Bohnen die gleiche Farbe haben, so ist eine schwarze Bohne in die Dose zurück zu legen
  (2) Falls die Bohnen verschiedene Farben haben, so ist nur die weiße Bohne zurück zu legen

Spielregeln als Semi-Thue-Regeln

_____

_____

Es sind zwei Spielverlauf-Beispiele bei einem Doseninhalt von vier weißen und drei schwarzen Bohnen anzugeben

_____

_____

Interaktion 38: Kaffeedosenspiel als Semi-Thue-System

Es ergeben sich für eine Eingabe – das ist bei diesem Spiel der am Anfang des Spiels bestehende Doseninhalt – ganz unterschiedliche Spielverläufe, wie am Beispiel einer Eingabe von vier weißen und drei schwarzen Bohnen in Interaktion 38 gezeigt werden soll. Die Spielverläufe gleichen sich allerdings sowohl in der Anzahl der Ableitungen (entspricht der

Anzahl der Spielrunden) und im Ergebnis (der Farbe der letzten Bohne in der Dose), wie in der folgenden Interaktion 39 näher auszuführen ist.

- Das Kaffeedosenspiel terminiert, wobei immer eine Bohne in der Dose bleibt

  Begründung:

  _____

  _____

- Das Ergebnis ist unabhängig vom Spielverlauf

  Begründung:

  _____

  _____

Interaktion 39: Eigenschaften des Kaffeedosenspiels

Die Terminierungseigenschaft des Kaffeedosenspiels kann man sich dadurch klarmachen, dass die stetige Abnahme der Anzahl der Bohnen in der Dose gewährleistet ist. Die Eigenschaft, dass das Ergebnis unabhängig vom Spielverlauf ist, lässt sich über eine Invariante erklären, die nach einer beliebigen Regelanwendung gültig bleibt.

3.4 Markov-Algorithmen

Die Eigenschaft des Nichtdeterminismus ist nur in Ausnahmefällen wünschenswert. Die Forderung nach deterministischen Semi-Thue-Systemen führt zu den Markov-Algorithmen. Andrei Markov (russischer Mathematiker) setzte dabei nicht auf den Arbeiten von Thue auf, sondern entwickelte diese von ihm selbst als normale Algorithmen bezeichneten Textersetzungssysteme parallel zu den Semi-Thue-Systemen Anfang der 50er Jahre.

- Ein (gesteuerter) Markov-Algorithmus ist ein deterministisches Semi-Thue-System, das zur Beschreibung von Textersetzungen dient

- Bestandteile eines Markov-Algorithmus
  - endlich viele Regeln
  - Einführung einer so genannten haltenden Regel $x \rightarrow .y$
  - Metaregeln:

    (MM1) Wähle in jedem Schritt die erste anwendbare Regel. Falls sie auf mehrere Teilwörter anwendbar ist, wende sie auf das am weitesten links stehende Teilwort an

    (MM2) Wende Regeln solange an, bis eine haltende Regel angewandt wurde, oder bis keine Regel mehr anwendbar ist

- Es werden zusätzliche Zeichen (α, β, χ, ...) verwendet, die weder im Eingabetext noch im Ergebnis vorkommen

Information 40: Markov-Algorithmen

Der Determinismus beim Markov-Algorithmus wird dadurch erreicht, dass die Regel-Anwendung gemäß der Reihenfolge der Aufschreibung erfolgt. Diese erste anwendbare Regel ist auf das am weitesten links stehende Teilwort anzuwenden.

Außerdem werden zwei verschiedene Formen von Endebedingungen vorgesehen. Neben der bereits von den Semi-Thue-Systemen bekannten Endebedingung, dass keine anwendbare Regel mehr existiert, wird außerdem eine spezielle haltende Regel eingeführt, die durch den Punkt nach dem Pfeil kenntlich gemacht ist.

Die zusätzlichen Zeichen, die in etwa den Hilfsvariablen zur Aufnahme von Zwischenergebnissen in der normalen Programmierung entsprechen, werden auch als "Schiffchen" bezeichnet. Die Assoziation ist dabei, dass die Hilfszeichen durch das Wort hindurch manövrieren und dadurch Kontextinformation zu speichern gestatten.

Die Arbeitsweise von Markov-Algorithmen soll anhand eines einfachen Beispiels verdeutlicht werden.

- Regeln
  (1) $\alpha L \rightarrow L\alpha$
  (2) $\alpha O \rightarrow O\alpha$
  (3) $\alpha \rightarrow ß$
  (4) $Lß \rightarrow ßO$
  (5) $Oß \rightarrow .\ L$
  (6) $ß \rightarrow .\ L$
  (7) $\varepsilon \rightarrow \alpha$

- Ablauf und Ergebnis des Algorithmus für das Wort LOLL

$$LOLL \Rightarrow \alpha LOLL \Rightarrow L\alpha OLL \Rightarrow LO\alpha LL \Rightarrow LOL\alpha L$$

$$\Rightarrow LOLL\alpha \Rightarrow LOLLß \Rightarrow LOLßO \Rightarrow LOß00 \Rightarrow LLOO$$

Information 41: Beispiel eines Markov-Algorithmus

Der Beispiel-Algorithmus besteht aus sieben Ersetzungsregeln. Die Nummerierung (die hier nur zur Veranschaulichung angegeben ist und fehlen kann) macht die Priorisierung deutlich (d.h., zuerst wird versucht, Regel (1) anzuwenden, dann Regel (2) usw.).

Das Ziel ist, eine Wirkungsweise (Semantik) mit den Regeln zu verbinden, die ja ansonsten nur ein rein syntaktisches Spiel darstellen.

Die Wirkungsweise kann an dem in Information 41 angegeben Beispiel nachvollzogen werden.

3.5 Formale Systeme

Semi-Thue-Systeme und deren deterministische Abwandlung in Form der Markov-Algorithmen sind Beispiele für formale Systeme. Formale Systeme beschreiben die allgemeinste Form von Relationen zwischen Gegenständen, die mit algorithmischen Methoden modelliert und realisiert werden können, wie in Information 42 näher ausgeführt wird.

- (U, $\Rightarrow$) ist ein formales System, wenn
  - U eine endliche oder abzählbar unendliche Menge ist
  - $\Rightarrow \subseteq U \times U$ eine Relation ist,
  - es einen Algorithmus gibt, der jedes $r \in U$ mit $l \Rightarrow r$, $l \in U$, in endlich vielen Schritten aus l berechnet
    - r heißt dann aus l (effektiv) berechenbar

- Bereits behandelte Beispiele formaler Systeme
  - U ist eine Menge von Wörtern
  - $\Rightarrow$ Ableitungsrelation eines Semi-Thue-Systems oder eines Markov-Algorithmus

- Ein anderes Beispiel
  - U $Z \times Z$ (Z: ganze Zahlen)
  - $\Rightarrow$ Addition (m, n) $\Rightarrow$ (m+n, 0)

Information 42: Formale Systeme

Zwei Elemente l und r aus der Menge U stehen genau dann in Relation, wenn es einen Algorithmus gibt, der l als Eingabe und r als Ausgabe hat. In diesem Falle gilt, dass r aus l effektiv berechenbar ist. Offensichtlich wird deshalb durch jedes Semi-Thue-System und jeden Markov-Algorithmus ein formales System definiert.

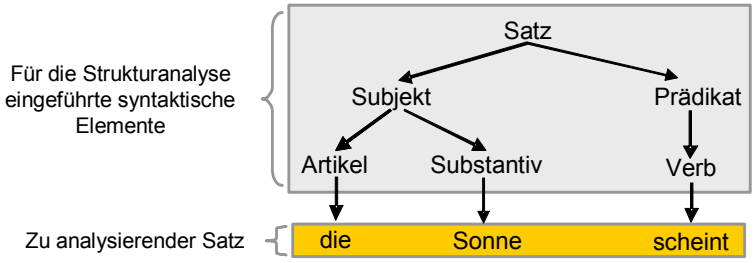
Es bestehen aber auch noch ganz andere Konstruktionsmöglichkeiten, wie das Beispiel einer Relation, die auf der Addition von zwei ganzen Zahlen basiert. Es sei angemerkt, dass in diesem Beispiel die ganzen Zahlen Z nicht gegen eine beliebige andere Zahlenmenge ausgetauscht werden darf. Die Konstruktion gilt auch für die natürlichen und rationalen Zahlen, nicht aber für die reellen Zahlen R, da diese überabzählbar sind.

4 GRAMMATIKEN

Die Linguisten haben in den 50er Jahren mithilfe von Semi-Thue-Systemen den Aufbau und die Struktur von natürlichen Sprachen analysiert und in Form von so genannten Chomsky-Grammatiken formal beschrieben. An erster Stelle ist hier Noam Chomsky (amerikanischer Linguist) zu nennen, nach dem auch die für diese Arbeiten erstellten Semi-Thue-Systeme, die Chomsky-Grammatiken, ernannt wurden [Go97].

www.cm-tm.uka.de/info1
Info-1-Team (Prof. Abeck)

- Grammatiken sind Ausprägungen von Semi-Thue-Systemen
  - Regeln heißen Produktionen
  - Ziel ist die Strukturanalyse und die Beschreibung von Sprachen



- Beispiele für Produktionen

Interaktion 43: GRAMMATIKEN - Spezielle Semi-Thue-Systeme

Das Beispiel in Interaktion 43 zeigt eine Analyse eines einfachen Satzes. Das Ergebnis ist ein Ableitungsbaum, der aus der Anwendung von gewissen Grammatikregeln hervorgeht. Die Regeln werden in den Grammatiken als Produktionen bezeichnet.

Das Ziel ist die Analyse und die Erzeugung (Produktion) von Sprachen. Den Ausgangspunkt der Analyse bilden die Bestandteile, aus denen ein Satz der (natürlichen) Sprache aufgebaut sein kann. Beispiele für solche Bestandteile, die als syntaktische Elemente bezeichnet werden, sind Satz, Subjekt, Prädikat, Artikel, Substantiv, Verb.

Aus dem obigen Ableitungsbaum sollen in Interaktion 43 einige Produktionen exemplarisch formuliert werden.

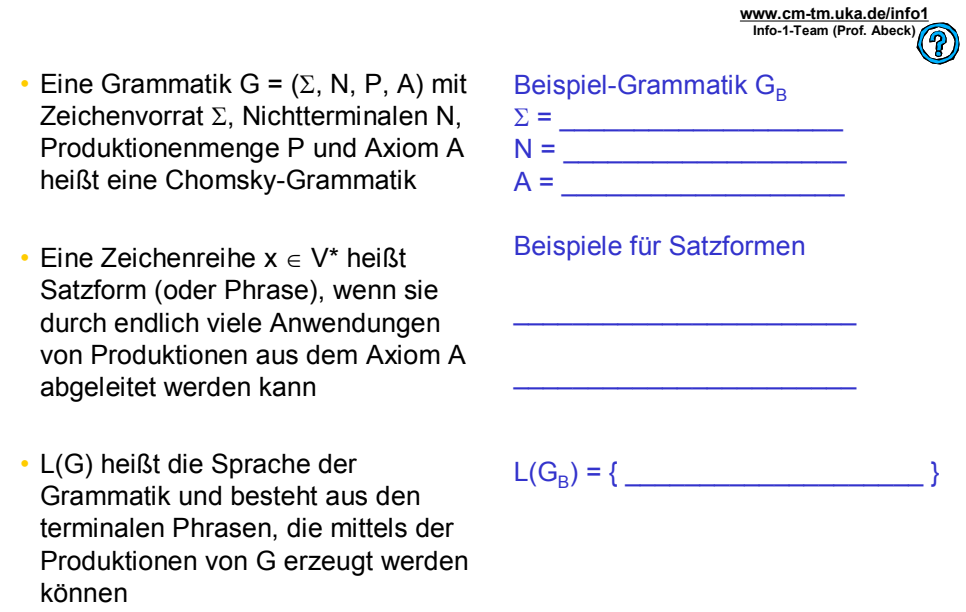
www.cm-tm.uka.de/info1
Info-1-Team (Prof. Abeck)

- In einer Grammatik werden zwei Arten von Zeichen unterschieden:
  1. Zeichenvorrat N der Nichtterminale
  2. Zeichenvorrat Σ der Terminale (Einzelzeichen der Sprache)
- $V = N \cup \Sigma$ heißt das Vokabular der Grammatik
- Ein ausgezeichnetes Element A aus N heißt das Axiom
  - bildet das Startsymbol
- Produktionen sind Regeln $l \rightarrow r$ mit Zeichenreihen $l, r \in V^*$

Information 44: Bestandteile einer Grammatik

Diese syntaktischen Elemente bilden in einer Grammatik den Zeichenvorrat der so genannten Nichtterminale (siehe Information 44). Dieser wird ergänzt um den Vorrat mit Zeichen, die konkret in den Sätzen der analysierten Sprache auftreten können. Da diese Zeichen nicht weiter in andere Zeichen überführt werden können, heißen sie Terminalzeichen. Nichtterminale und Terminale bilden gemeinsam das Vokabular der Grammatik. Ein ausgezeichnetes Nichtterminal, das so genannte Axiom bildet das Startsymbol.

Eine Produktion besteht aus einer linken Seite l und einer rechten Seite r. Ihnen liegt das allgemeine Prinzip der Textersetzung zugrunde.



Interaktion 45: Grammatik und Sprache

Eine Chomsky-Grammatik ist formal als ein 4-Tupel aufzufassen mit den vier Elementen Terminale, Nichtterminale, Produktionen und Axiom.
In Interaktion 43 wurden die Produktionen der in Interaktion 45 als G_B bezeichneten Grammatik bereits teilweise erfasst, weshalb jetzt noch zur Vervollständigung der 4-Tupel-Beschreibung die Zeichenvorräte und das Axiom anzugeben sind.
Satzformen, die Nichtterminale beinhalten – also keine terminalen Phrasen darstellen – sind gewissermaßen die Zwischenergebnisse einer vom Axiom startenden Ableitung. Die in G_B auftretenden Satzformen und die von dieser Grammatik erzeugte Sprache sind zu ermitteln.

4.1 Chomsky-Hierarchie

Welche Sprache durch eine Grammatik erzeugt werden kann, hängt davon ab, welche Anforderungen an den Aufbau der Produktionen gestellt wird. Je nach Anforderung werden gemäß einer Chomsky-Hierarchie die in Information 46 aufgeführten Grammatiktypen unterscheiden.

- Chomsky-0-Grammatik (CH-0)
  - allgemeiner Produktionstyp: $l \rightarrow r$ wobei $l, r \in V^*$ beliebig
    - insbesondere auch ε-Produktionen ($r = \varepsilon$)

- Chomsky-1-Grammatik (CH-1)
  - längenbeschränkt: $l \rightarrow r$ wobei $l, r \in V^*$, $1 \leq |l| \leq |r|$
  - kontextsensitiv: $uAv \rightarrow urv$ wobei $A \in N$, $u, v \in V^*$
    $r \in V^+$, d.h. $r \neq \varepsilon$

  Anmerkung: Zu jeder CH-1-Grammatik kann die Produktion
  $S \rightarrow \varepsilon$ wobei S das Axiom ist
  hinzugefügt werden

- Chomsky-2-Grammatik (CH-2)
  - kontextfrei: $A \rightarrow r$ wobei $A \in N$, $r \in V^*$

- Chomsky-3-Grammatik (CH-3)
  - linkslinear: $A \rightarrow Bx$ oder $A \rightarrow x$ wobei $A, B \in N$, $x \in \Sigma$
  - rechtslinear: $A \rightarrow xB$ oder $A \rightarrow x$ wobei $A, B \in N$, $x \in \Sigma$

Information 46: Chomsky-Grammatiktypen

Die Chomsky-Grammatiken werden in Abhängigkeit der erlaubten Typen von Produktionen in insgesamt vier Typen (Typ 0 bis Typ 3, CH-0 bis CH-3) eingeteilt. Dabei bilden die vier Typen eine Hierarchie in der Form, dass eine mittels einer CH-i-Grammatik erzeugte Sprache auch mittels einer CH-i-1-Grammatik erzeugt werden kann – aber nicht zwingend umgekehrt. D.h., CH-3-Sprachen sind in CH-2-Sprachen enthalten, CH-2-Sprachen in CH-1-Sprachen und CH-1-Sprachen in CH-0-Sprachen.

Bei CH-0 ist das leere Wort ε auf beiden Seiten einer Regel erlaubt, d.h.
- ε-Produktionen (rechte Seite $r = \varepsilon$, also $l \rightarrow \varepsilon$)
- Produktionen, die aus dem leeren Wort ε etwas produzieren, also $\varepsilon \rightarrow r$.

Man kann jede z.B. durch Markov-Algorithmen berechenbare Menge als Sprache einer CH-0-Grammatik erhalten, was bedeutet, dass CH-0 genauso mächtig ist wie Markov-Algorithmen (oder auch wie die nichtdeterministischen Semi-Thue-Systeme).

Für CH-1 gibt es zwei Festlegungen, die die gleiche Einschränkung bedeuten. CH-1-Grammatiken heißen wegen der zweiten Art von Produktions-Restriktion auch kontextsensitive Grammatiken, wobei die „Kontexte“ hier die beiden Zeichenreihen u und v darstellen.
Die eine Richtung, nämlich dass kontextsensitive Produktionen zugleich längenbeschränkt sind, ist offensichtlich, weil $r \neq \varepsilon$ gefordert ist.
Durch die zulässige Ergänzung einer CH-1-Grammatik um die Produktion $S \rightarrow \varepsilon$ wird sichergestellt, dass zu jeder CH-2-Sprache eine CH-1-Grammatik angegeben werden kann.

CH-2-Grammatiken heißen kontextfrei, weil die Produktionen ohne Vorhandensein eines Kontextes angewendet werden können.
Wie die Definition in Information 46 zeigt, gilt $r \in V^*$, was $r = \varepsilon$ einschließt. Eine kontextfreie Grammatik, die auf ε-Produktionen und Produktionen der Form $V \rightarrow V$ verzichtet (also auf linker und rechter Seite jeweils ein Nichtterminalzeichen) heißt anständig (*proper*).
Die Menge der kontextfreien Sprachen ist erheblich weniger umfangreich als die Menge der kontextsensitiven Sprachen. Allerdings kann man mit kontextfreien Grammatiken sehr viel leichter umgehen, weshalb alle gängigen Programmiersprachen, wie z.B. Java oder C, auf der Basis von CH-2-Grammatiken beschrieben sind.

Noch eingeschränkter sind die Möglichkeiten, die die CH-3-Grammatiken bieten, die auch als reguläre Grammatiken bezeichnet werden. Produktionen, die auf der rechten Seite nur ein terminales Zeichen aufweisen, heißen terminierend.

- Beispielgrammatik $G_A = (\Sigma, N, P, A)$
  - Σ = {bez, +, *, (,)}
  - N = {A, T, F}
  - P = { A $\rightarrow$ T | A + T,
    T $\rightarrow$ F | T * F,
    F $\rightarrow$ bez | (A) }
  - A ist das Axiom
- Beispiel einer Ableitung einer Formel aus dem Axiom A

$$
\begin{aligned}
A &\Rightarrow A + T \Rightarrow T + T \Rightarrow T * F + T \\
&\Rightarrow F * F + T \Rightarrow (A) * F + T \\
&\Rightarrow (A + T) * F + T \Rightarrow (T + T) * F + T \\
&\Rightarrow (F + T) * F + T \Rightarrow (bez + T) * F + T \\
&\Rightarrow (bez + F) * F + T \Rightarrow (bez + bez) * F + T \\
&\Rightarrow (bez + bez) * bez + T \\
&\Rightarrow (bez + bez) * bez + F \\
&\Rightarrow (bez + bez) * bez + bez
\end{aligned}
$$

Ableitungsbaum zu
(bez + bez) * bez + bez

Interaktion 47: Beispiel einer kontextfreien Grammatik

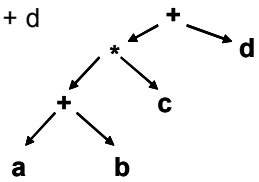
Die in Interaktion 47 angegebene Grammatik G_A beschreibt den Aufbau arithmetischer Ausdrücke mit den Operatoren + und *. Die Grammatik unterscheidet die drei Nichtterminale A (steht für Ausdruck und ist zugleich das Axiom), T (Term) und F (Faktor). Das terminale Zeichen bez (Bezeichner) steht für beliebige einfache Operanden (Variablen, Konstante). Die Zeichen + und * repräsentieren die bekannten Operatoren. Die Klammerzeichen (und) dienen zur Beeinflussung der Reihenfolge der Operator-Auswertung.

An einem einfachen Beispiel-Ausdruck (bez + bez) * bez + bez soll gezeigt werden, wie sich die Produktionen anwenden lassen. Die Ableitungsfolge soll in Interaktion 47 graphisch durch einen so genannten Ableitungsbaum beschrieben werden, wodurch sich die Anwendung der Produktionen sehr viel anschaulicher darstellen lässt.

Der Übergang vom Ableitungsbaum zum bereits kennen gelernten Kantorowitsch-Baum wird durch eine Eliminierung der Nichtterminale (d.h. eine Beschränkung ausschließlich auf die Terminalzeichen) erreicht.

- Ein Ableitungsbaum eines arithmetischen Ausdrucks, der auf das Wesentliche beschränkt ist, heißt Kantorowitsch-Baum

- Kantorowitsch-Baum zum vorhergehenden Beispiel
  A ⇒* (a + b) * c + d



- Ableitung: A ⇒ A + A ⇒ A * A + A ⇒ (A) * A + A ⇒ (A+ A) * A + A ⇒ (bez + A) * A + A ⇒ (bez + bez) * A + A ⇒ (bez + bez) * bez + A ⇒ (bez + bez) * bez + bez

- Wie lautet die dazugehörige (vereinfachte) Grammatik?

Interaktion 48: Grammatik und Kantorowitsch-Baum

Das Prinzip besteht darin, anstelle des Nichtterminals das Operatorzeichen an der Stelle im Ableitungsbaum vorzusehen. Aus der in Interaktion 48 gesuchten Grammatik geht hervor, dass es zu jedem Operator (hier + und *) jeweils genau eine Produktion gibt.

Im Kantorowitsch-Baum fehlen die Klammersymbole. Diese werden nicht explizit benötigt, weil sich die notwendige Klammersetzung aus der Struktur des Baumes und den Vorrangregeln der Operatoren ergibt.

4.2 Backus-Naur-Form

Zur Beschreibung von höheren Programmiersprachen wurde eine praxistaugliche Notation für kontextfreie Grammatiken gesucht. Dieses Ziel wurde von John Backus 1959 mit einer Notation verfolgt, die von Peter Naur das erste Mal zur Beschreibung von Algol 60 genutzt wurde und daher heute als Backus-Naur-Form (BNF) bezeichnet wird.

Die BNF hat eine hohe praktische Bedeutung, da sie auch heute noch zur Beschreibung der Syntax von Programmiersprachen dient. In der Kurseinheit PROGRAMMIERGRUNDLAGEN wird eine entsprechend erweiterte Form der BNF zur Beschreibung der Java-Syntax genutzt [Mö03].

- Kontextfreie Sprachen werden zur Beschreibung der Syntax von Programmiersprachen verwendet

- John Backus gab hierzu folgende Notation an (wurde von Peter Naur erstmals zur Beschreibung von Algol 60 eingesetzt):
  - Nichtterminale sind sprechende Wörter und werden in spitzen Klammern geschrieben
    - Variante: es wird auf die spitzen Klammern verzichtet und stattdessen werden die Terminale in Apostroph ".." gesetzt
  - statt $\rightarrow$ benutzte Backus das Symbol ::=
    - heute wird statt ::= häufig nur das Gleicheitszeichen = verwendet
  - Der senkrechte Strich | ermöglicht die Angabe alternativer rechter Seiten zu einer linken Seite
    - diese Konvention ist erhalten geblieben

Information 49: Backus-Naur-Form (BNF) und Anpassungen

Die Notation wurde im Laufe der Zeit weiterentwickelt, wie in Information 49 beschrieben wird. Der wesentliche Unterschied in der Aufschreibung betrifft die Konvention, wie Terminale und Nichtterminale unterschieden werden. In der weiterentwickelten BNF-Notation werden nicht die Nichtterminale (für die Backus eckige Klammern vorgeschlagen hat), sondern die Terminale durch Apostroph-Zeichen gekennzeichnet.

- Die Beispiel-Grammatik mit

$P = \{ \; A \rightarrow T \mid A + T,$
$\quad\quad T \rightarrow F \mid T * F,$
$\quad\quad F \rightarrow bez \mid (A) \quad \}$

in (angepasster) BNF-Notation:

```
Ausdruck = Term | Ausdruck "+" Term.
```

Interaktion 50: BNF-Beispiel

In Interaktion 50 ist die BNF-Notation zur Aufschreibung der Produktionen der Beispiel-Grammatik zu nutzen. Aus der Nutzung der BNF zur Beschreibung von Programmiersprachen ergab sich die Anforderung, gewisse Metazeichen einzuführen, durch die die Grammatik kompakter und verständlicher dargestellt werden konnte.

- Die BNF wurde um so genannte Metazeichen zu einer EBNF-Notation erweitert
  - es ist zu beachten, dass diese Metazeichen nicht verbindlich festgelegt sind und daher unterschiedliche Notationen existieren

- Die folgende Notation wird hier verwendet
  - = trennt linke und rechte Regelseite
  - . schließt Regel ab
  - | trennt Alternativen
    Beispiel: x | y beschreibt: x, y
  - () klammert Alternativen
    Beispiel: (x | y) z beschreibt: xz, yz
  - [] wahlweises Vorkommen
    Beispiel: [x] y beschreibt: xy, y
  - {} 0-maliges bis n-maliges Vorkommen
    Beispiel: {x} y beschreibt: y, xy, xxy, ...

Information 51: Erweiterte Backus-Naur-Form (EBNF)

Es bestehen verschiedene Ausprägungen von EBNFs, die die Erweiterungen unter Verwendung unterschiedlicher Metazeichen beschreiben. In Information 51 sind die in der Kurseinheit PROGRAMMIERGRUNDLAGEN ausführlich behandelten Metazeichen aufgeführt.

5 ENDLICHE AUTOMATEN

Ein endlicher Automat ist ein Beispiel einer (abstrakten) Maschine. Es besteht ein enger Zusammenhang zwischen Grammatiken (bzw. den von diesem erzeugten formalen Sprachen) und bestimmten Arten von Maschinen [Go97].

- Ein endlicher Automat ist eine Art von (abstrakter) Maschine

- Es besteht ein enger Zusammenhang zwischen den durch Grammatiken erzeugten Sprachen und den Maschinen
  - Maschine soll zu einer als Eingabe anliegenden Zeichenreihe feststellen, ob diese Zeichenreihe zur Sprache gehört oder nicht

- Zu jedem Chomsky-Grammatiktyp lässt sich ein diese Sprachklasse bearbeitbarer Maschinentyp angeben
  - CH-0 Turing-Maschine
  - CH-1 Linear beschränkter Automat
  - CH-2 Kellerautomat
  - CH-3 Endlicher Automat

- Eine vertiefte Behandlung des Zusammenhangs zwischen Sprachen und Maschinen ist Teil der Berechenbarkeitstheorie

Information 52: ENDLICHE AUTOMATEN - Einordnung

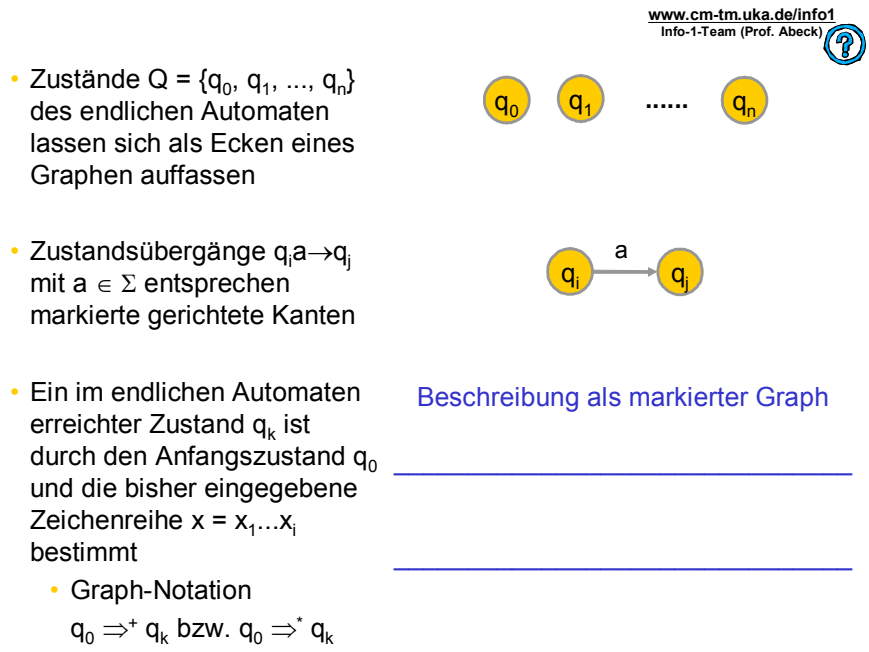
Den Zusammenhang zwischen den Chomsky-Grammatik und den in der Informatik unterschiedenen Maschinentypen zeigt Information 52. Der in diesem Kapitel näher behandelte Endliche Automat ist dabei der "schwächste" Maschinentyp, dessen Mächtigkeit sich allerdings durch gewisse Verallgemeinerungen erhöhen lässt. Viele Systeme, die zu einer Eingabe eine definierte Ausgabe erzeugen, lassen sich mithilfe von Automaten analysieren, beschreiben und realisieren.

- Ein endlicher Automat wird in der Informatik zur Analyse, Beschreibung und Realisierung von Systemen eingesetzt

- Bestandteile und Arbeitsweise eines endlichen Automaten
  - endliche Menge Q von Zuständen mit einem Anfangszustand $q_0 \in Q$
  - Zeichenvorrat Σ
  - Lesen eines Zeichens $a \in \Sigma$ führt zu einem Zustandsübergang vom aktuellen Zustand $q \in Q$ in einen neuen Zustand $q' \in Q$
    - Notation: $qa \rightarrow q'$
    - bei gegebenem q bestimmt a den Nachfolgerzustand bzw. bei gegebenem a hängt die Wirkung q´ vom bisherigen Zustand q ab
    - der Zustand läßt sich als ein (endliches) Gedächtnis über die Vorgeschichte und die bisher eingegebenen Zeichen auffassen

Information 53: Einsatz eines endlichen Automaten

Das Automatenmodell kann dabei an verschiedenen Stellen erweitert werden, wodurch der Einsatzbereich dieses Modells erheblich vergrößert wird. Zum Automatenmodell gehören eine Zustandsmenge Q, ein Zeichenvorrat Σ und eine Menge von Zustandsübergängen.



Interaktion 54: Endliche Automaten und Graphen

Es besteht eine enge Verbindung zwischen Automaten und Graphen. Ein endlicher Automat kann als gerichteter Graph mit den Zuständen Q als Eckenmenge und den Zustandsübergängen als markierte Kanten angesehen werden (siehe Interaktion 54). Falls mittels einer Zeichenreihe $x = x_1...x_i$ ein Übergang vom Ausgangszustand q_0 in einen Zustand q_k vollzogen werden kann, so beschreibt diese Zeichenreihe den Weg von q_0 nach q_k im Graphen. Zu beachten ist, dass die durchwanderten Zustände dabei nicht zwingend verschieden sein müssen.
Entsprechend zu übernehmen ist die eingeführte Graph-Notation $q_0 \Rightarrow^+ q$ bzw. $q_0 \Rightarrow^* q$, falls auch das leere Eingabewort $x = \varepsilon$ zugelassen ist.

Die Automaten-Arten unterscheiden sich darin, auf welche Weise sie Ausgaben erzeugen.

- Mealy-Automat
  - Erzeugung einer Ausgabe bei jedem Zustandsübergang
  - Ausgabe ist ein Wort $t = t_0...t_{n-1}$ über einem Ausgabezeichenvorrat T
  - Markieren der Kanten mit a / t
- Moore-Automat
  - Erzeugung einer Ausgabe bei Erreichen eines Zustands
  - Ausgabe ist ein Wort $t \in T^n$
- Akzeptor
  - häufigster Spezialfall eines Moore-Automaten
  - Ausgabe nicht bei allen Zuständen
  - Zustände $F \in Q$, bei denen eine Ausgabe erfolgt, heißen Endzustände
  - Ausgegebenes Wort $t \in T^n$ hängt vom erreichten Endzustand $q \in F$ ab

Information 55: Arten von Automaten

Gemäß diesem Kriterium werden in Information 55 drei Arten von Automaten unterschieden:

(1) **Mealy-Automat**
In der Notation a / t bedeutet a das Zeichen, durch das der Übergang erfolgt (Übergangszeichen) und $t = t_0...t_{n-1}$ das beim Übergang erzeugte Ausgabewort.

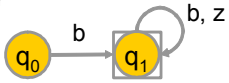
(2) **Moore-Automat**
Welche Ausgabe erfolgt, hängt nicht vom Zustandsübergang, sondern vom Zustand ab. Das bedeutet, dass beim Moore-Automaten die Ausgabe unabhängig davon ist, über welchen Übergang ein Zustand erreicht wurde.

(3) **Akzeptor**
Der Akzeptor ist die in der Informatik am häufigsten zum Einsatz kommende Automaten-Art. Das Neue am Akzeptor ist die Auszeichnung von gewissen Zuständen, bei denen der Automat eine Ausgabe erzeugt. Diese Zustände heißen Endzustände.

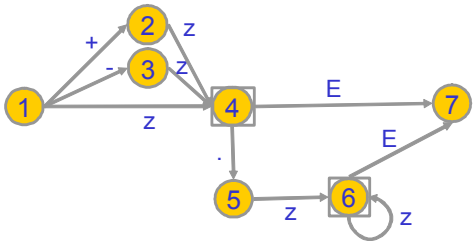
5.1 Akzeptoren

Endliche Automaten werden in der Informatik u.a. im Zusammenhang mit der Übersetzung von höheren Programmiersprachen (Übersetzerbau) genutzt. Warum die endlichen Automaten bzw. die Akzeptoren hier so wichtig sind, verdeutlichen die in Interaktion 56 angegebenen zwei Beispiele.

- Erkennen von Bezeichnern einer üblichen Programmiersprache
  - Aufbau eines Bezeichners: erstes Zeichen ist ein Buchstabe gefolgt von beliebigen Buchstaben oder Ziffern
    - b: beliebiger Buchstabe aus {a, ..., z, A, ..., Z}
    - z: beliebige Ziffer aus {0, ...,9}
  - Übergangsgraph des Bezeichner-Akzeptors:



- Erkennen einer Gleitkommazahl einer üblichen Programmiersprache
  - Beispiele: 3 67.34 5E2 2.58E-15
  - Zeichenvorrat {z, E, ., +, -}, wobei z eine beliebige Ziffer ist
  - Übergangsgraph des Gleitkommazahl-Akzeptors (zu vervollständigen)



Interaktion 56: Beispiele für Akzeptoren

Das erste Beispiel beschreibt einen Akzeptor, durch den Bezeichner, wie sie in einer üblichen höheren Programmiersprache verwendet werden, erkannt werden.

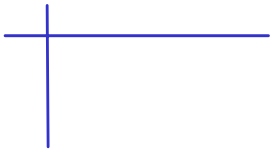
Buchstabe ist dabei ein Zeichen aus der Menge {a,..., z, A,...,Z} und Ziffer ist ein Zeichen aus der Menge {0,...,9}. Es werden Platzhalter für Buchstaben in Form eines b und für Ziffern im Form eines z eingeführt. Der Übergangsgraph eines Akzeptors, der eine beliebige Zeichenfolge daraufhin überprüft, ob diese ein Bezeichner ist, besteht aus zwei Zuständen q_0 und q_1, wobei q_1 der Endzustand ist.

Diese Art von Akzeptor wird es in jedem Übersetzerprogramm (Compiler) einer Programmiersprache geben. Es besteht folgende zusätzliche Anforderung

- Bezeichner sind nicht nur als solche zu erkennen, sondern sind außerdem in so genannte Symboltabellen abzuspeichern.
- Der Akzeptor ist somit zugleich ein Mealy-Automat, der bekanntlich die Möglichkeit bietet, die eingegebenen Zeichen zu verarbeiten, in diesem Falle in eine Tabelle einzutragen.

Das zweite Beispiel (Gleitkommazahl-Akzeptor) zeigt, dass gemäß diesem Prinzip in einfacher und übersichtlicher Form komplexere Strukturen beschrieben werden können. Der Akzeptor verarbeitet eine beliebig aufgebaute Gleitkommazahl, wie diese üblicherweise in höheren Programmiersprachen auftreten. Der Anteil des Akzeptors, der den Exponentanteil (nach dem Exponentsymbol E) folgt, ist in der folgenden Interaktion zu ergänzen.

www.cm-tm.uka.de/info1
Info-1-Team (Prof. Abeck)



- Ein Automat heißt ein vollständiger Akzeptor, wenn die Übergangsmatrix vollständig um Übergänge in einen Fehlerzustand ergänzt ist
  - es ist die vollständige Übergangstabelle für den Bezeichner-Akzeptor anzugeben
- Ein endlicher Akzeptor lässt sich als Quintupel (Σ, Q, q_0, F, P) auffassen
  - Σ : Zeichenvorrat
  - Q: nichtleere endliche Zustandsmenge
  - q_0: Anfangszustand aus Q
  - F: nichtleere Menge von Endzuständen aus Q
  - P: Übergänge $q_a \rightarrow q'$ mit $q, q' \in Q, a \in \Sigma$
- Sprache, die der Akzeptor akzeptiert:
  $L(A) = \{x \mid x \in \Sigma^*, q_0 x \Rightarrow^* q_e, q_e \in F\}$

Interaktion 57: Festlegungen zum Akzeptor

Ein vollständiger Akzeptor ist dadurch ausgezeichnet, dass er die Fehlerfälle explizit und vollständig in der Übergangsmartix behandelt. In Interaktion 57 wird nach der vollständigen Übergangsmatrix des Bezeichner-Akzeptors gefragt. Die wesentliche Idee, die dem vollständigen Akzeptor zugrunde liegt, besteht in der Einführung eines Fehlerzustandes und die Ergänzung sämtlicher Fehlerübergänge in diesen Zustand.

Formal kann ein endlicher Akzeptor als ein aus fünf Elementen aufgebautes System, also als Quintupel angesehen werden. Drei der fünf Elemente sind Mengen, und zwar eine Menge von (1) Zeichen, (2) Zuständen und (3) Übergängen, die wie in Grammatiken als Produktionen bezeichnet werden.
Die Endlichkeits-Eigenschaft des Akzeptors bezieht sich auf die (als endlich geforderte) Zustandsmenge.

Die zwei noch nicht erwähnten Elemente des Quintupels hängen auch unmittelbar mit der Zustandsmenge zusammen. Aus der Zustandsmenge wird ein mit $q_0 \in Q$ bezeichneter Anfangszustand (auch Startzustand genannt) und eine Menge von Endzuständen $F \subset Q$ festgelegt.

Die Sprache, die einem Akzeptor zugeordnet werden kann, hängt eng mit dem Anfangs- und Endzustand zusammen, da sie aus allen Zeichenreihen besteht, die einen Weg vom Anfangszustand zu einem Endzustand beschreiben. Formal lässt sich die Sprache wie in Interaktion 57 angegeben ist, definieren.

Im Folgenden wird eine Eigenschaft der Übergänge bzw. Produktionen eines Automaten näher betrachtet. Die Übergänge lassen sich auffassen als eine Abbildung $\delta: Q \times \Sigma \rightarrow Q$.

- Bei den bisherigen Beispielen
  - die Übergänge $q_a \rightarrow q'$ stellen eine Abbildung δ: $Q \times \Sigma \rightarrow Q$ dar
  - ein endlicher Automat, der die Eigenschaft besitzt, dass die Übergänge zusammen genommen eine (Übergangs-) Funktion bilden, heißt deterministisch

- Im allgemeinen Fall wird der Nichtdeterminismus zugelassen
  - die Übergänge definieren ein durch P endlich erzeugtes Semi-Thue-System mit Eingabezeichenvorrat $\sum$ und syntaktischen Hilfszeichen $q \in Q$
  - die Auffassung eines endlichen Automaten als Semi-Thue-System stellt eine Beziehung zu den regulären Chomsky-Grammatiken her

Information 58: Determinismus und Nichtdeterminismus

Die hier interessierende Frage ist, ob zu einem Paar (Zustand, Zeichen) nur ein oder etwa mehrere Folgezustände bestehen.

Der nichtdeterministische Fall ist der allgemeine Fall, der unmittelbar zu den Semi-Thue-Systemen führt. Bei der Auffassung eines endlichen Automaten als Semi-Thue-System spielen die Zustände die Rolle der syntaktischen Hilfszeichen. Zustandsinformation und syntaktische Hilfszeichen sind beides Formen von Kontextinformation, durch die der Berechnungsvorgang „gesteuert“ wird.
Es kann ein Übergang zwischen endlichen Automaten und Semi-Thue-Systemen konstruktiv hergestellt werden.

5.2 Regulärer Ausdruck

Neben den endlichen Automaten und den Chomsky-3-Grammatiken existiert eine dritte Beschreibungsform, die diese Klasse von Sprachen auszudrücken gestattet: die so genannten Regulären Ausdrücke.

- Ein regulärer Ausdruck R über einem Zeichenvorrat C ist induktiv definiert durch:

  (1) c ist ein regulärer Ausdruck für jedes $c \in C$

  (2) Ist R ein regulärer Ausdruck, dann auch (R)*

  (3) Sind R, S reguläre Ausdrücke, so sind auch (RS) und (R + S) reguläre Ausdrücke

- Beispiele
  - Bezeichner: b(b+z)*
  - Ganze Zahlen: zz*
  - Dezimalbrüche: zz* ’/ ’ zz*

Information 59: Regulärer Ausdruck

Wie Information 59 verdeutlicht, sind die regulären Ausdrücke induktiv definiert, was heißt, dass man mit einem elementaren regulären Ausdruck beginnt und nachfolgend aus einem oder mehreren bereits bestehenden regulären Ausdrücken einen neuen regulären Ausdruck konstruiert.

Jede der oben angegebenen Konstruktionsvorschriften erweitert die Menge von Wörtern M(R), die mit dem regulären Ausdruck R verknüpft ist.

Der Vorteil der regulären Ausdrücke besteht darin, dass sie eine kompakte und trotzdem gut lesbare Beschreibung der Sprache ermöglichen, wie die Beispiele in Information 59 zeigen.

www.cm-tm.uka.de/info1
Info-1-Team (Prof. Abeck)

- Die im Zusammenhang mit regulären Ausdrücken definierten Operationen erfüllen verschiedene Gesetze
- Eine Menge L* ist genau dann Sprache einer regulären Grammatik G, wenn L durch einen regulären Ausdruck beschrieben wird
- Durch die folgenden Schritte [1] bis [4] ist ein regulärer Ausdruck R in einen endlichen Akzeptor überführbar

  [1] Füge zu Beginn von R sowie nach jedem terminalen Zeichen eine Zahl ein

  [2] Einzelnes Zeichen bedeutet Zustandsübergang

  [3] Vereinigung führt zu Zustandsübergängen der zu Beginn gegebenen Zustände in alle durch Einzelzeichen erreichbaren Folgezustände

  [4] Kleenescher Stern S* führt zu Zustandsübergängen aus sämtlichen Endzuständen von S in die aus den Anfangszuständen von S mit einem Zeichen erreichbaren Zustände

Information 60: Eigenschaften regulärer Ausdrücke

Reguläre Ausdrücke besitzen die interessante Eigenschaft, dass mit ihnen „gerechnet“ werden kann. Die Grundlage hierfür wird durch entsprechende Rechenregeln geschaffen, wie sie vom Rechnen mit Zahlen her bekannt sind. Es ist zu beachten, dass die Gesetze jeweils nur für gewisse Operationen gelten. So gilt z.B. das Kommutativgesetz für die Vereinigung, nicht aber für die Konkatenation.

Der in Information 60 angegebene Satz besagt, dass reguläre Ausdrücke die gleiche Mächtigkeit wie reguläre Grammatiken (und damit auch wie endliche Automaten) haben.

In den Beweisschritten, die einen regulären Ausdruck in eine reguläre Grammatik überführen, ist ein konstruktives Vorgehen enthalten, das gewissermaßen eine „Bauanleitung“ für den endlichen Akzeptor darstellt, der die durch den regulären Ausdruck beschriebene Sprache akzeptiert. In Information 60 ist das konstruktive Vorgehen skizziert.

Die CH3-Sprachklasse lässt sich einfach und effizient durch einen Rechner verarbeiten. Diese positive Eigenschaft trifft auch für die CH2-Sprachklasse zu, nicht allerdings für CH1 und CH0.

Eine vertiefte Behandlung von Formalen Sprachen und deren vielfältigen Einsatzformen innerhalb der Informatik (z.B. im Übersetzerbau) erfolgt in verschiedenen weiterführenden Veranstaltungen im Rahmen des Informatikstudiums.

VERZEICHNISSE

Abkürzungen und Glossar

Abkürzung oder Begriff	Langbezeichnung und/oder Begriffserklärung
Adjazenzliste	Darstellungsform eines Graphen, bei der zu jeder Ecke alle diejenigen aufgelistet, zu denen eine Kante existiert.
Adjazenzmatrix	Eine Matrix, durch die ein Graph repräsentiert wird. Die Zeilen- und Spaltenzahl entspricht dabei der Eckenanzahl des Graphen. Die Zeilen und Spalten der Matrix sind mit den Nummern der (zuvor geeignet durchnummerierten) Ecken indiziert. Der Wert eines Adjazenzmatrix-Elements (i, j) bestimmt, ob eine Kante zwischen Ecke i und Ecke j besteht (Wert 1) oder nicht (Wert 0).
Algebra	Ein Tripel bestehend aus (Trägermenge, Operationen, Gesetze).
Algebraische Abgeschlossenheit	Das Ergebnis der Anwendung einer Operation der Algebra auf Elemente der Trägermenge liefert wieder ein Element der Trägermenge.
Axiom	Bezeichnung des in einer Grammatik ausgezeichneten Nichtterminals, das die Wurzel jedes Ableitungsbaumes bildet. Synonymer Begriff: Startsymbol
Baum (ungerichtet)	Graph, der die Eigenschaften der Zyklenfreiheit und des Zusammenhängens erfüllt.
Boolesche Algebra	Eine für die Informatik besonders wichtige Algebra. Es handelt sich hierbei um einen vollständigen, komplementären, distributiven Verband.
BNF	Backus-Naur-Form Notation für kontextfreie Grammatiken (CH2) zur Beschreibung von höheren Programmiersprachen.
Chomsky-Grammatik	Spezielles Semi-Thue-System, durch das der Aufbau und die Struktur von natürlichen und künstlichen Sprachen analysiert und formal beschrieben werden kann.
Chomsky-Hierarchie	Sprachklassen, die aus Bildungsgesetzen von Grammatik-Produktionen entstehen und eine hierarchische Anordnung der formalen Sprachen bilden.
Dualitätsprinzip	Prinzip, dass in einer Algebra gilt, falls jedes ihrer Gesetz doppelt auftritt. Das duale Gesetz erhält man durch Ersetzen einer Operation gegen ihr duales Gegenüber.
Endlicher Automat	Beispiel einer (abstrakten) Maschine, durch die formale Sprachen der CH3-Sprachklasse behandelt werden können.
Einselement	Ein Element ε, für das gilt:

	(1) $\varepsilon \times a = a$ (2) $a = a \times \varepsilon$ Synonymer Begriff: Neutrales Element
Formales System	Die allgemeinste Form der Beschreibung von Relationen zwischen Gegenständen, die mit algorithmischen Methoden modelliert und realisiert werden können.
Grad	Eigenschaft einer Ecke eines Graphen, die die Anzahl der von der Ecke ausgehen bzw. zu der Ecke führenden Kanten angibt.
Graph	Anschauliche Darstellung einer Relation in Form von Ecken und den bestehenden Beziehungen zwischen den Ecken. Die Begriffe 'Relation' und 'Graph' werden in der Informatik weitgehend synonym verwendet.
Halbgruppe	Eine spezielle Algebra mit der Konkatenation als einziger Operation, in der das Assoziativgesetz gilt.
Kantorowitsch-Baum	Baum, durch der den Aufbau eines Terms in Teilterme dargestellt wird.
Konkatenation	Bezeichnung einer Operation, durch die zwei Zeichenketten verknüpft werden. Synonymer Begriff: Verkettung
Metaregel	Regel, die festlegt, in welcher Reihenfolge und an welcher Stelle eine Menge von (Textersetzungs-) Regeln anzuwenden sind.
Monoid	Halbgruppe, in der es ein Einselement gibt.
Nichtdeterminiertheit	Eigenschaft des Algorithmus-Ergebnisses, das für eine Eingabe unterschiedlich sein kann (d.h., das Ergebnis ist nicht vorhersagbar).
Nichtdeterminismus	Eigenschaft des Algorithmus-Verhaltens, das für eine Eingabe unterschiedlich sein (d.h., das Verhalten ist nicht vorhersagbar).
Nichtterminierung	Eigenschaft eines Algorithmus, der für eine Eingabe endlos läuft und somit keine (d.h., eine undefinierte) Ausgabe liefert.
Reguläre Ausdrücke	In der Praxis häufig genutzte Form zur Beschreibung einer zur CH3-Sprachklasse gehörenden regulären Sprache.
Relation	Liefert den grundlegenden Formalismus, Beziehungen zwischen Gegenständen zu beschreiben.
Schlinge	Ein Zyklus der Länge 1.
Semi-Thue-System	System, das einen aus einem vorgegebenen endlichen Zeichenvorrat gebildeten Eingabetext mittels (endlicher oder abzählbar undendlicher) Textersetzungsregeln in einen Ausgabe Text überführt. Semi-Thue-Systeme arbeiten nichtdeterministisch. Synonymer Begriff: Textersetzungssystem Englischer Begriff: *String Replacement System*, *String Rewrite System*

UML	*Unified Modeling Language* Sprache, die aus graphischen Elemente zur semi-formalen Beschreibung von beliebigen Gegenständen (z.B. Software-Systeme oder Geschäftsbereiche) besteht.
Wald	Graph, der die Eigenschaft der Zyklenfreiheit, nicht aber die Eigenschaft des Zusammenhängens erfüllt.
Weg	Ein Kantenzug mit einer Folge von Kanten, bei denen die Eingangsecke der einen Kante mit der Ausgangsecke der anderen Kante übereinstimmen.
Zyklus	Ein spezieller Weg, der an der Ecke endet, an der er beginnt. Synonymer Begriff: Kreis

Index

Informationen und Interaktionen

Literatur

[Br98] Manfred Broy: Informatik – Eine grundlegende Einführung, Band 1: Programmierung und Rechenstrukturen, Springer Verlag 1998.

[Go97] Gerhard Goos: Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer Verlag 1997.

[Mö03] Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

[Sa04] Gunter Saake, Kai-Uwe Sattler: Algorithmen und Datenstrukturen – Eine Einführung mit Java, dpunkt.verlag, 2004.

[Se02] Robert Sedgewick: Algorithmen, Addison-Wesley – Pearson Studium, 2002.

RECHENSTRUKTUREN UND FUNTKIONALE PROGRAMME

Kurzbeschreibung

Rechenstrukturen sind das Informatik-Gegenstück zur Algebra und führen zu den Termen und Termersetzungssystemen. Es werden die Terme der Aussagen- und der Prädikatenlogik behandelt. Anschließend werden die Funktionsapplikation und die Rekursion als die zentralen Elemente funktionaler Programme vorgestellt und am Beispiel des rekursiven Java-Methodenaufrufs veranschaulicht.

Schlüsselwörter

Algebra, Rechenstruktur, Signatur, Natürliche Zahlen, Ganze Zahlen, Boolesche Algebra, NAT, BOOL, Term, Grundterm, Termersetzung, Aussagenlogik, Prädikatenlogik, Quantoren, Funktionale Programmierung, Applikative Programmierung, Funktionsanwendung, Funktionsapplikation, Funktionsabstraktion, Konversionen, Rekursion, Rekursiver Java-Methodenaufruf, Nachklappern, Iteration

Lernziele

1. Die Bedeutung von Rechenstrukturen als ein zentrales Verbindungselement zwischen der Mathematik und der Informatik wird verstanden.
2. Der Umgang mit Termen und mit Systemen zur Termersetzung wird beherrscht.
3. Die Grundlagen der mathematischen Logik und der daraus hervorgehenden Rechenstrukturen der Aussagenlogik und der Prädikatenlogik können nachvollzogen werden.
4. Der Zusammenhang der funktionalen Programmierung zu den Rechenstrukturen wird verstanden.
5. Die Sprachelemente der funktionalen Programmierung sind bekannt und können innerhalb der Programmierung genutzt werden.
6. Das Prinzip der Rekursion wird verstanden und rekursive Java-Programme können erstellt werden.

Hauptquellen

- Manfred Broy: Informatik – Eine grundlegende Einführung, Band 1: Programmierung und Rechenstrukturen, Springer Verlag 1998.
- Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

Inhaltsverzeichnis

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- AUFBAU UND BEISPIELE VON RECHENSTRUKTUREN
  - Signatur, Operationen, Kantorowitsch-Baum
- TERMERSETZUNG
  - Termersetzungsregel, Termersetzungssystem, Algorithmus, Korrektheit
- MATHEMATISCHE LOGIK
  - Formel, Aussagenlogik, Prädikatenlogik
- FUNKTIONALE PROGRAMMIERKONZEPTE
  - Sprachelemente, Syntax und Semantik, Termersetzungskonzept, Bedingte Ausdrücke, Funktionsanwendung, Funktionsabstraktion
- REKURSIVE FUNKTIONSDEKLARATION
  - Rekursiver Java-Methodenaufruf, Nachklappern, Beispiele add(), mult(), mod(), fact()

Information 1: RECHENSTRUKTUREN UND FUNKTIONALE PROGRAMME

1 AUFBAU UND BEISPIELE VON RECHEN-STRUKTUREN

Die Rechenstruktur ist das Gebilde, das die Algebra und die darin festgelegten Strukturen in der Informatik verankert. Mittels Rechenstrukturen wird somit ein Übergang von der Algebra hin zu den Algorithmen und Programmen geschaffen [Br98].

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Algorithmen
  - arbeiten über Datenelementen, den Trägermengen
  - bestehen aus Operationen auf den Trägermengen
- Trägermengen und Operationen lassen sich zu Rechenstrukturen zusammenfassen
  - entsprechen dem Begriff der Algebra
- Beispiele, die als Rechenstruktur formuliert werden können
  - Taschenrechner
  - leistungsstarke Rechenanlage
- Definition: Rechenstruktur
  Seien S und F Mengen von Bezeichnungen. Eine Rechenstruktur A besteht aus einer Familie $\{s^A: s \in S\}$ von Trägermengen s^A und einer Familie $\{f^A: f \in F\}$ von Abbildungen f^A zwischen diesen Trägermengen.

Information 2: AUFBAU UND BEISPIELE VON RECHENSTRUKTUREN – Definition einer Rechenstruktur

Die Operationen, die in einem Algorithmus genutzt werden, hängen unmittelbar mit den Trägermengen zusammen, da sie Abbildungen zwischen diesen schaffen. Für die Begriffe der Operationen bzw. der Abbildungen benutzen Mathematiker und Informatiker auch beide den Begriff der Funktion.

Das Gebilde der Rechenstruktur ist äußerst mächtig. Es ermöglicht die Beschreibung beliebig komplexer Sachverhalte und Systeme. So lassen sich einfachere Systeme wie ein Taschenrechner genauso mittels einer Rechenstruktur beschreiben wie auch das komplexeste Rechensystem.

In Information 2 werden eine formale Definition einer Rechenstruktur und eine gewisse Notation eingeführt. Die Rechenstruktur A wird auch als Tupel $A = (\{s^A: s \in S\}, \{f^A: f \in F\})$ geschrieben.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Sorten S sind Bezeichnungen für Trägermengen
- Die Elemente $f \in F$ heißen Funktionssymbole oder Operationssymbole
- Einführung eines speziellen Elements $\perp$ (“Bottom”-Element)
  - repräsentiert nicht definierte Funktionswerte, wodurch partielle Abbildungen vermieden werden
  - eine Menge M, die nicht $\perp$ enthält, wird um das Bottom-Element ergänzt

$$M^{\perp} =_{def} M \cup \{\perp\}$$

- Eine Funktion $f: M^{\perp}_1 \times M^{\perp}_2 \times \ldots \times M^{\perp}_n \rightarrow M^{\perp}_{n+1}$ heißt strikt, wenn gilt:
  Ist eines der Argumente $\perp$, so ist auch das Resultat der Funktion $\perp$

Information 3: Strikte Funktion

Die Trägermengen-Elemente s werden auch als Sorten bezeichnet, die Funktionselemente f als Funktionssymbole. Es gilt folgender Zusammenhang zwischen Sorten und Funktionen innerhalb einer Rechenstruktur: Zu einer n-stelligen Funktion f^A existieren Sorten $s_1, \ldots, s_{n+1} \in S$, so dass gilt: $f^A: s_1^A \times \ldots \times s_n^A \rightarrow s_{n+1}^A$

Eine Funktion einer Rechenstruktur kann an gewissen Stellen kein Ergebnis liefern, also an dieser Stelle undefiniert sein. Zu diesem Zweck wird das Zeichen $\perp$ ("Bottom") eingeführt, dessen Semantik „undefinierter Wert“ ist. Auf diese Problematik wird auch im Kontext mit der Programmierung in der Kurseinheit IMPERATIVE PROGRAMMIERUNG eingegangen.

Formal lässt sich eine beliebige Trägermenge M um dieses Zeichen ergänzen und entsprechend durch ein hochgestelltes $\perp$ anzeigen. Angewendet auf Rechenstrukturen können dadurch partielle Funktionen vermieden werden.

Im Zusammenhang mit der Einführung des Zeichens $\perp$ steht eine Eigenschaft von Rechenstrukturen, die als strikt bezeichnet wird. Diese Eigenschaft stellt eine bestimmte Anforderung an Argumente und das Ergebnis im Hinblick auf das Undefiniert-Zeichen, nämlich dass aus der Undefiniertheit von nur einem Argument die Undefiniertheit des Ergebnisses folgt.

1.1 Definition einer Rechenstruktur

Die Definition einer Rechenstruktur geschieht in der Weise, dass in einem ersten Schritt die drei relevanten Mengen (Sorten, Funktionssymbole und Trägermengen) festgelegt werden und anschließend die Funktionssymbole näher beschrieben werden.

- Menge S der Sorten: $S = \{bool\}$
- Menge F der Funktionssymbole: $F = \{true, false, \neg, \vee, \wedge\}$
- Zuordnung Sorten zu Trägermengen $bool^{Bool} = \mathcal{B}^{\perp}$
- Zuordnung von Funktionen zu den Funktionssymbolen

$(a, b \in \mathcal{B})$		Stelligkeit	Schreibweise
$true^{BOOL}: \rightarrow \mathcal{B}^{\perp}$	$true^{BOOL} = L,$	_____	_____
$false^{BOOL}: \rightarrow \mathcal{B}^{\perp}$	$false^{BOOL} = O,$	_____	_____
$\neg^{BOOL}: \mathcal{B}^{\perp} \rightarrow \mathcal{B}^{\perp}$	$\neg^{BOOL} b = \complement b,$	_____	_____
$a \vee^{BOOL} b : \mathcal{B}^{\perp} \times \mathcal{B}^{\perp} \rightarrow \mathcal{B}^{\perp}$	$a \vee^{BOOL} b = a \vee b$	_____	_____
$a \wedge^{BOOL} b : \mathcal{B}^{\perp} \times \mathcal{B}^{\perp} \rightarrow \mathcal{B}^{\perp}$	$a \wedge^{BOOL} b = a \wedge b$	_____	_____

- Die in der Rechenstruktur BOOL festgelegten Funktionen sind strikt

Interaktion 4: Rechenstruktur BOOL der booleschen Werte

Die zwei Schritte sind somit:

- **Festlegung der Sorten, der Funktionssymbole und der Trägermengen der Rechenstruktur**
  Die Schreibweise $bool^{BOOL}$ ist eine konkrete Ausprägung der eingeführten Notation s^A zur Festlegung der Trägermengen von Rechenstrukturen. Mit $bool^{BOOL} = \mathcal{B}^{\perp}$ wird der Zusammenhang zur früher eingeführten Booleschen Algebra $\mathcal{B}$ hergestellt.
- **Beschreibung der Funktionssymbole durch Festelegung der Definitions- und Wertemenge und Zuordnung der Funktionen**
  Es werden die im Zusammenhang mit der booleschen Algebra $\mathcal{B}$ auftretenden Funktionssymbole genutzt. Die Stelligkeit und die Schreibweise (Präfix, Infix, Postfix) ist in Interaktion 4 zu jeder der in der Rechenstruktur BOOL gehörenden Funktionen anzugeben. Die Funktionen erfüllen die Striktheitseigenschaft, d.h. sie liefern als Ergebnis undefiniert, sobald nur eines der Argumentwerte undefiniert ist.

- Menge S der Sorten: S = {bool, nat}
- Menge F der Funktionssymbole: F = {true, false, $\neg$, $\wedge$, $\vee$, zero, succ, pred, add, mult, sub, div, $\leq$, $\doteq$}
- Zuordnung Sorten zu Trägermengen: $bool^{NAT} = \mathcal{B}^{\perp}$, $nat^{NAT} = \mathbb{N}^{\perp}$
- Zuordnung von Funktionen zu den Funktionssymbolen (x, y $\in \mathbb{N}$):

$zero^{NAT}: \rightarrow \mathbb{N}^{\perp}$	$zero^{NAT} = 0$,
$succ^{NAT}: \mathbb{N}^{\perp} \rightarrow \mathbb{N}^{\perp}$	$succ^{NAT}(x) = x+1$,
$pred^{NAT}: \mathbb{N}^{\perp} \rightarrow \mathbb{N}^{\perp}$	$pred^{NAT}(x) = x-1$, falls $x \geq 1$
	$pred^{NAT}(0) = \perp$,
$add^{NAT}: \mathbb{N}^{\perp} \times \mathbb{N}^{\perp} \rightarrow \mathbb{N}^{\perp}$	$add^{NAT}(x, y) = x+y$
$mult^{NAT}: \mathbb{N}^{\perp} \times \mathbb{N}^{\perp} \rightarrow \mathbb{N}^{\perp}$	$mult^{NAT}(x, y) = x*y$
$sub^{NAT}: \mathbb{N}^{\perp} \times \mathbb{N}^{\perp} \rightarrow \mathbb{N}^{\perp}$	$sub^{NAT}(x, y) = x-y$,
$div^{NAT}: \mathbb{N}^{\perp} \times \mathbb{N}^{\perp} \rightarrow \mathbb{N}^{\perp}$	$div^{NAT}(x, y) = x \div y$, falls $y > 0$
	$div^{NAT}(x, 0) = \perp$,
$\leq^{NAT}: \mathbb{N}^{\perp} \times \mathbb{N}^{\perp} \rightarrow \mathcal{B}^{\perp}$	$0 \leq^{NAT} x = L$, $(x+1 \leq^{NAT} 0) = O$
	$(x+1 \leq^{NAT} y+1) = (x \leq^{NAT} y)$,
$\doteq^{NAT}: \mathbb{N}^{\perp} \times \mathbb{N}^{\perp} \rightarrow \mathcal{B}^{\perp}$	$(x \doteq^{NAT} y) = (x \leq^{NAT} y) \wedge^{NAT} (y \leq^{NAT} x)$

Information 5: Rechenstruktur NAT der natürlichen Zahlen

Die Rechenstruktur NAT benutzt die Sorte bool und fügt dieser die Sorte nat hinzu. Die Sorte bool wird im Zusammenhang mit den letzten zwei Funktionssymbolen $\leq$ und $\doteq$ benötigt, da diese einen booleschen Wahrheitswert als Ergebnis liefern.

Die in der Rechenstruktur auftretenden Funktionssymbole stützen sich auf die bekannten Operatoren, die auf natürlichen Zahlen angewendet werden können.

Bei der Operation pred, die den Vorgänger (*Predecessor*) zum Eingabeargument liefert, muss der Bereich, in dem die Funktion definiert ist, explizit bei der Spezifikation angegeben werden, indem dem Wert 0 der undefinierte Wert $\perp$ als Vorgänger zugeordnet wird. Entsprechendes gilt für die Division durch 0.

Durch die in Information 5 auf der linken Seite angegebenen Gleichungen wird die Wirkungsweise und somit die Semantik einer Operation beschrieben. Die Operationen werden auf entsprechende Funktionen der genutzten Algebren zurückgeführt. Das gilt z.B. für die 0-stellige Operation zero genauso wie für die 1-stellige Operation succ oder die 2-stellige Operation add.

Für alle Funktionen gilt: Falls nur ein Eingabeargument $\perp$ (also undefiniert) ist, so ist auch das Ergebnis $\perp$.

1.2 Signatur und Signaturdiagramm

Durch eine Signatur wird festgelegt, in welcher Weise die Funktionssymbole mit den Sorten sinnvoll verknüpft werden können.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

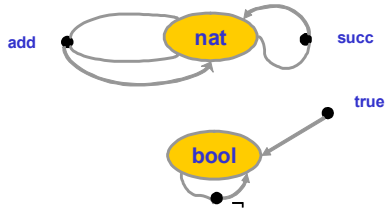
- In Anlehnung an den Signatur-Begriff von Algebren wird die Signatur zu einer Rechenvorschrift als ein Paar (S, F) von Mengen S und F eingeführt
  - S wird als die Menge der Sorten bezeichnet
  - F wird als die Menge der Funktionssymbole bezeichnet
- fct f $\in S^+$ beschreibt die Funktionalität zu jedem f $\in$ F
- Schreibweise: fct f = $(s_1, \ldots, s_n)\ s_{n+1}$
  - entspricht der mathematischen Schreibweise
    f^A: $s_1^A \times \cdots \times s_n^A \rightarrow s_{n+1}^A$

Information 6: Signatur einer Rechenvorschrift

Eine Funktion ist dabei Element eines über den Sorten gebildeten n-Tupels. Das in Information 6 verwendete Symbol S^+ drückt aus, dass $n \geq 1$ ist. Im Falle von $n = 1$ handelt es sich um eine 0-stellige (konstante) Funktion. Die Funktionsschreibweise entspricht der angegebenen mathematischen Schreibweise.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Beispiel: Signatur der Rechenstruktur NAT
  - S_{NAT} = {bool, nat}
  - F_{NAT} = {true, false, $\neg$, $\wedge$, $\vee$, zero, succ, pred, add, mult, sub, div, $\leq$, $\doteq$}
  - fct true = bool fct $\neg$ = (bool) bool
    fct succ = (nat) nat fct add = (nat, nat) nat
    ...
- Signaturen lassen sich graphisch in Form von Signaturdiagrammen übersichtlich darstellen
  - Beispiel: Signaturdiagramm zur Rechenstruktur NAT (zu ergänzen)



Interaktion 7: Signatur und Signaturdiagramm zur Rechenstruktur NAT

Die textuelle Beschreibung einer Signatur lässt sich mittels eines Signaturdiagramms in eine graphische Darstellung überführen. Es handelt sich hierbei um einen speziellen Graphen mit zwei Arten von Ecken zur Repräsentation der an der Rechenstruktur beteiligten Sorten und Funktionen. Weiterführende Beschreibungen zu Graphen finden sich in der Kurseinheit ALGEBRAISCHE STRUKTUREN UND FORMALE SYSTEME.

Die Funktionalitäten zu jeder beteiligten Funktion werden durch die Kanten dargestellt, wie in Interaktion 7 am Beispiel von 0-, 1- und 2-stelligen Funktionen verdeutlicht wird. Die in dieser

Interaktion gestellte Aufgabe besteht darin, das Signaturdiagramm gemäß der Signatur von NAT zu vervollständigen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Die Signatur INT der ganzen Zahlen stimmt bis auf Umbenennung von Sorten mit der Signatur NAT der natürlichen Zahlen überein
  - S = {bool, int}
    $int^{INT} = \mathbb{Z}$
  - F = {succ, pred, zero, ...}
    - $pred^{INT}(z) = z-1$
      - für alle $z \in \mathbb{Z}$
      - beachte: $pred^{NAT}(x) = x - 1$, falls $x \geq 1$; $pred^{NAT}(0) = \perp$
    - übrige Funktionen wie Rechenstruktur NAT
      - die Gleichungen, durch die die Wirkungsweise beschrieben ist, sind entsprechend anzupassen

- Die Signaturen von NAT und INT machen deutlich, dass sich mit der gleichen Signatur unterschiedliche Rechenstrukturen verbinden lassen
  - eine Signatur charakterisiert eine Rechenstruktur nicht eindeutig

Information 8: Rechenstruktur der ganzen Zahlen INT

Die für die natürlichen Zahlen NAT erstellte Signatur und das dazugehörige Signaturdiagramm lassen sich ohne strukturelle Änderungen – d.h. bis auf Umbenennungen – zur Beschreibung der Rechenstruktur der ganzen Zahlen INT verwenden. NAT und INT besitzen die gleichen syntaktischen Strukturen.

Der einzige (nicht an der Signatur erkennbare) Unterschied tritt bei der Vorgängerfunktion pred auf. Das Eingabeargument muss mehr überprüft werden, da jede ganze Zahl einen Vorgänger hat. Die Bedingungen, wie sie für die Vorgängerfunktion im Zusammenhang mit den natürlichen Zahlen definiert wurde, treten somit bei den ganzen Zahlen nicht mehr auf.

Die korrekten aus einer Signatur ableitbaren Ausdrücke werden als Terme bezeichnet. Im Folgenden werden die Terme genauer betrachtet, weil hieraus eine für die Informatik besonders wichtige Struktur, die Termalgebra, resultiert.

1.3 Grundterme und Termalgebra

Die Menge der Grundterme zu allen Sorten einer Signatur Σ wird mit W_Σ bezeichnet.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Zu einer Signatur $\Sigma = (S, F)$ ist die Menge der Grundterme W_Σ^s der Sorte s mit $s \in S$ definiert durch
  - (1) jedes nullstellige Funktionssymbol $f \in F$ mit fct f = s bildet einen Grundterm der Sorte s
  - (2) jede Zeichenreihe $f(t_1, \ldots, t_n)$ mit $f \in F$ und fct f = $(s_1, \ldots, s_n)$ s ist ein Grundterm der Sorte s, falls für alle i, $1 \leq i \leq n$, t_i ein Grundterm der Sorte s_i ist

- Es ist ein Grundterm zur Signatur der Rechenstruktur NAT zu formulieren, in dem die Funktionen

  zero, true, ¬, succ, add

  vorkommen

  _____

- Welche Schreibweise wurde benutzt? _____

Interaktion 9: Grundterme

Die Menge W_Σ zerfällt in disjunkte Teilmengen W_Σ^s der Sorte s mit $s \in S$. In Interaktion 9 wird das Bildungsgesetz eines Grundterms beschrieben. Gemäß diesem Bildungsgesetz ist ein die angegebenen Anforderungen erfüllender Grundterm zu formulieren und die Bezeichnung der dabei verwendeten Schreibweise ist anzugeben.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Grundterme lassen sich in einer Rechenstruktur A mit Signatur Σ interpretieren
  - $I^A: W_\Sigma \rightarrow \{a \in s^A: s \in S\}$
  - $I^A[t]$ (oder kurz t^A) bezeichnet die Interpretation des Grundterms t in A
  - $I^A[f(t_1,\ldots, t_n)] = f^A(I^A[t_1], \ldots, I^A[t_n]) = f^A(t_1^A, \ldots, t_n^A)$

- Beispiel: Interpretation von Grundtermen über der Signatur NAT
  - I^{NAT} [pred(succ(zero))] = _____

    _____

    _____
  - I^{NAT}[pred(zero)] = _____

Interaktion 10: Interpretation von Grundtermen

Beim Grundterm t wird davon ausgegangen, dass es sich um eine Repräsentation der Sorte s handelt, wobei s ein Element der Sortenmenge S der Rechenstruktur A ist.

In den in Interaktion 10 angegebenen Beispielen ist das Vorgehen der Interpretation auf die zwei Gundterme pred(succ(zero)) und pred(zero) über der Signatur NAT anzuwenden. Durch die Interpretation I wird ein Grundterm der Sorte nat auf ein Element der Trägermenge abgebildet.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Eine Rechenstruktur A heißt termerzeugt, wenn für jedes Element a der Trägermengen von A eine Termrepräsentation existiert
  - d.h. ein Grundterm der Sorte s mit $t^A = a$
  - die Interpretationsabbildung ist in diesem Fall surjektiv
- Ist die Rechenstruktur NAT termerzeugt? ja ☐ nein ☐
- Falls ja, ist eine Termrepräsentation anzugeben

_____

_____

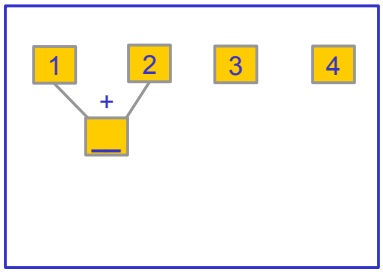
Interaktion 11: Termerzeugte Rechenstruktur

Eine Rechenstruktur wird als termerzeugt bezeichnet, wenn es für jedes Element a einer Trägermenge (s^A) einen Grundterm zur Sorte s gibt, der durch Interpretation auf dieses Element abgebildet wird ($t^A = a$). Diese Eigenschaft besagt, dass das Bild der Interpretationsfunktion, also die Trägermenge vollständig erfasst wird und somit kein Element der Trägermenge ausgespart wird. Die Anforderung wird von den surjektiven Abbildungen erfüllt.
In Interaktion 11 wird die Aufgabe gestellt, eine solche surjektive Abbildung für die Rechenstruktur NAT zu finden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Die Interpretation, also der Wert eines Grundterms, lässt sich entsprechend seiner Termstruktur ausrechnen
- Geeignetes Hilfsmittel zur Berechnung sind Formulare
  - graphische Darstellung für die Berechnung der Interpretation
  - Einführung von Rechtecken, die jeweils die Interpretation eines Teilterms beinhalten
  - Verknüpfen der Rechtecke durch Kanten entlang des Termaufbaus
- Beispiel: Grundterm ((1 + 2) · 3) - 4

  Das Formular ist zu vervollständigen



Interaktion 12: Formulare

Mit der Interpretations-Abbildung ist eine Wertezuordnung verbunden. Es stellt sich die Frage, wie man diese Wertezuordnung durch ein systematisches Berechnungsprinzip durchführen kann. Eine Antwort liefern die in Interaktion 12 eingeführten Formulare.

Ein Formular ist formal betrachtet ein Baum, also ein Graph ohne Zyklen mit einer ausgezeichneten Wurzel. Die Knoten werden als Rechtecke gezeichnet und repräsentieren entsprechende Teilterme bzw. Wurzeln von Teilbäumen, die ihrerseits Teilterme repräsentieren.

1.4 Identifikatoren

Bei den zuvor beispielhaft behandelten Grundtermen waren die in den Termen auftretenden Trägermengenelemente mit festem Wert vorgegeben. Nur deshalb konnte man die Berechnung im Formular durchführen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Identifikatoren sind Platzhalter für Terme (oder Elemente), die später an der entsprechenden Stelle eingesetzt werden können
  - Identifikatoren heißen auch Bezeichner, Variable oder Unbekannte
  - sie können als Namen für erst später genau bezeichnete Terme (oder Elemente) verstanden werden

- Ergänzung der Signatur $\Sigma = (S, F)$ um eine Menge $X = \{X_s: s \in S\}$ von Identifikatoren
  - führt zu einer um X erweiterten Termalgebra $W_\Sigma(X)$
  - wird auch als $W_{\Sigma'}$ bezeichnet mit $\Sigma' = (S, F \cup \{x \in X_s: s \in S\})$ und fct x = s für $x \in X_s$

- Beispiele:
  - Gleichung mit Unbekannten z.B. $ax^2 + bx + c = 0$
  - Funktionsdefinition z.B. $f: \mathbb{N} \to \mathbb{N}$ mit $f(x) = 2x + 1$

Information 13: Terme mit (freien) Identifikatoren

Häufig wünscht man sich bezüglich der Zuordnung von Werten zu den im Term auftretenden Elementen mehr Flexibilität. Das führt zu den Identifikatoren, die Platzhalter für Terme oder Elemente darstellen.

Formal bedeutet die Einführung von Identifikatoren eine Ergänzung der Signatur um eine Menge von Identifikatoren X_s der Sorte s. Die resultierende Termalgebra wird als $W_{\Sigma'}$ bezeichnet. Zu einer Funktion lassen sich dann entsprechend ein oder mehrere Identifikatoren hinzufügen.

Ein Beispiel einer Funktion mit freiem Identifikator ist die in Information 13 angegebene Gleichung mit Unbekannten. Terme mit freien Identifikatoren werden auch als Polynome bezeichnet. Das zweite Beispiel zeigt einen Term 2x +1 innerhalb einer Funktionsdefinition.

- Identifikatoren in Termen können durch andere Terme ersetzt werden
  - diese Abbildung wird als Substitution bezeichnet
  - t[r/x] bezeichnet den Term, der sich ergibt, wenn der Identifikator x in t durch r ersetzt wird

- Die Substitution ist induktiv über den Aufbau der Terme beschrieben
  - x[t/x] = __
  - y[t/x] = __ Annahme: y und x sind verschiedene Identifikatoren
  - $f(t_1, ..., t_n)[t/x] = f(t_1[t/x], ..., t_n[t/x])$
    - wobei $f \in F$ mit fct $f = (s_1, ..., s_n)\ s_{n+1}$
    - Terme t_i haben Sorten s_i

- Ein Term r heißt Instanz des Terms t, wenn r durch Substitution gewisser (freier) Identifikatoren aus t erhältlich ist

Interaktion 14: Substitution in Termen

Durch die Substitution wird ein formaler Mechanismus geschaffen, um mit den Identifikatoren zu arbeiten, insbesondere um diese mit Werten zu belegen.

Die Formulierung „induktiv über den Termaufbau“ besagt, dass eine Definition entlang der Bildungsgesetze, die für den Aufbau eines Terms bestehen, erfolgt. Die zwei zur Definition der Substitution genutzten Bildungsgesetze wurden in einem der vorherigen Abschnitte in Interaktion 9 angegeben.

Da der Termaufbau mit einfachen Identifikatoren beginnt, geht die Beschreibung der Substitution ebenfalls zunächst von Identifikatoren, hier x und y, aus (siehe Interaktion 14):

- x[t/x] bezeichnet dabei den Term, der sich ergibt, wenn x im "Term" x durch den Term t ersetzt wird. Da es sich bei diesem Term um einen einzelnen Identifikator handelt, wird das x gegen t ersetzt, d.h. das Ergebnis ist t.
- Falls es sich um einen Identifikator handelt, der von x verschieden ist, dann gäbe es kein zu substituierendes x und die Substitution würde gar nichts bewirken.
- Es besteht im Zusammenhang mit der Substitution noch eine weitere spezielle Schreibweise, die zugleich die dritte Regel ist, durch die die Substitution („induktiv über den Termaufbau“) beschrieben wird: $t[t_1/x_1, ..., t_n/x_n]$ bezeichnet den Term, der durch die simultane Substitution der Identifikatoren x_i durch die t_i aus t entsteht. Hierbei muss gefordert werden, dass die Identifikatoren x_i paarweise disjunkt sind.

Ein Term, der durch Substitution aus einem Term hervorgeht, heißt Instanz zu diesem Term. Das Vorgehen der Substitution und der Instanzbegriff werden im Folgenden an einem Beispiel veranschaulicht.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Term t sei definiert als mult(add(succ(x), y), z)
  - Der Identifikator x ist mit dem Wert 0, y mit 1 und z mit 2 zu belegen

_____

_____

  - Gesucht ist eine Instanz zu t

_____

Interaktion 15: Beispiel einer Substitution

Durch Interaktion 15 wird verdeutlicht, dass die Substitution dazu genutzt werden kann, in einem Term auftretende freie Variablen mit Werten zu belegen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Belegung von X in A
  - Vorgehen: jedem Identifikator x in X der Sorte s wird ein Element a der Trägermenge s^A zur Sorte s zugeordnet
  - Formal: $\beta:\{x \in X_s: s \in S\} \rightarrow \{a \in s^A: s \in S\}$
- Die Interpretation eines belegten Terms ist durch folgende zwei Gleichungen definiert
  (1) $I_\beta^A[x] = \beta(x)$
  (2) $I_\beta^A[f(t_1, ..., t_n)] = f^A(I_\beta^A[t_1], ..., I_\beta^A[t_n])$
- Punktweise Änderung von Belegungen

$$ß[m/x](z) = \begin{cases} m & \text{falls } z = x \\ ß(z) & \text{falls } z \neq x \end{cases}$$

Information 16: Interpretation von Termen

In Information 16 ist A eine Rechenstruktur mit $\Sigma = (S, F)$ und X eine Familie von Mengen von Identifikatoren. Der Begriff "Familie" besagt, dass zu jeder einzelnen Sorte $s \in S$ eine (Teil-) Menge von Identifikatoren X_S existiert. Diese Teilmengen X_S von Identifikatoren werden zur Festlegung einer Abbildung benötigt, die Belegung genannt wird und mit ß bezeichnet ist.

Gesucht ist die Interpretation $I_\beta^A[t]$ eines Terms t mit freien Identifikatoren aus X. Falls t lediglich ein Identifikator x aus X ist, so ist die Interpretation von t die Belegung von x.
Gemäß den Gesetzen des Termaufbaus ist ein komplexerer Term aufgebaut aus einer in der Funktionenmenge F der Signatur $\Sigma = (S, F)$ der Rechenstruktur A enthaltenen Funktion f und entsprechenden Termen t_1 bis t_n. In diesem Fall sind diese Terme zu interpretieren und die Ergebnisse sind als Argumente der Funktion f^A zuzuführen.

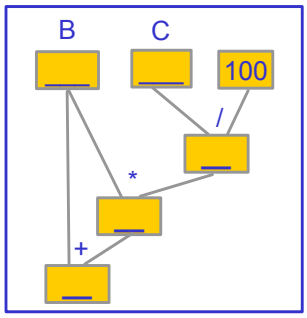
Häufig soll eine Belegung nur an einem Punkt abgeändert und an allen anderen Punkten übernommen werden. Hierfür dient eine in Information 16 angegebene Notation, die an die Substitution angelehnt ist.

- Terme mit freien Identifikatoren können als Rechenformulare gedeutet werden, in denen noch nicht alle Werte festgelegt sind
- Beispiel: A = B + B * (C/100)

  Was wird berechnet?

  _____

  _____



- Ein Term mit freien Identifikatoren definiert ein so genanntes Berechnungsschema
  - zyklenfreier gerichteter Graph, falls gewisse Identifikatoren an mehren Stellen im Term auftreten
  - Baum, falls jeder Identifikator nur an einer Stelle im Term auftritt

Interaktion 17: Terme mit freien Identifikatoren als Formulare

In Interaktion 17 werden nochmals die Formulare aufgegriffen, da sich diese zur Behandlung von Termen mit freien Identifikatoren besonders gut eignen. Die freien Identifikatoren stellen im Formular Felder dar, deren Werte im Zusammenhang mit der Belegung festgelegt werden müssen. Anhand des Terms A = B + B * (C / 100) wird der Sachverhalt verdeutlicht.

Ein solches Formular mit freien Identifikatoren wird als Berechnungsschema bezeichnet. Das Berechnungsformular entspricht dem auch in der Kurseinheit ALGEBRAISCHE STRUKTUREN UND FORMALE SYSTEME behandelten Kantorowitsch-Baum.
Komplexere Berechnungsschemata findet man u.a. im Verwaltungsbereich, wie z.B. in der Lohnsteuerberechnung. Ein Berechnungsschema weist genau dann eine Baumform auf, wenn Identifikatoren nur genau einmal auftreten. Tritt ein Identifikator mehrmals auf, so ergibt sich ein zyklenfreier gerichteter Graph, der auch als Hasse-Diagramm bezeichnet wird.

Die Rechenstrukturen bilden die formale Grundlage für abstrakten Datentypen, die zentraler Bestandteil der Programmierung sind und in der Kurseinheit OBJEKTORIENTIERTE PROGRAMMIERUNG im Detail behandelt werden.

2 TERMERSETZUNG

Durch die Terme und die darauf aufbauenden Termersetzungssysteme wird eine im Vergleich zu den in der Kurseinheit ALGEBRAISCHE STRUKTUREN UND FORMALE SYSTEME behandelten Textersetzungssystemen klarere und aussagekräftigere Möglichkeit zur Beschreibung von Algorithmen bereitgestellt [Br98].

- Die Beschreibung von Algorithmen als Textersetzungssysteme wurde bereits behandelt
- Die Ersetzung von Termen anstelle einzelner Zeichen führt zu einer klareren Beschreibungsmöglichkeit von Algorithmen
  - Terme lassen sich über gegebenen Signaturen nach festen Regeln aufbauen
    - Interpretation über einer Rechenstruktur
    - semantische Äquivalenz auf Termen
- Aufstellen von Termersetzungsregeln, die die semantische Äquivalenz berücksichtigen
  - Ausnutzen der speziellen Struktur der Terme, die sich aus ihrem Aufbau ergibt

Information 18: TERMERSETZUNG - Motivation

Der Weg von den Textersetzungssystemen zu den Termersetzungssystemen führt über die Festlegung, was ein Term ist, d.h. welche Syntax und Semantik ihm zugrunde gelegt werden soll.

Die Syntax von Termen ist durch den Termaufbau gegeben, der durch die Signatur und die zwei im letzten Kapitel in Interaktion 9 eingeführten Regeln festgelegt ist. Die Regeln wurden bereits an verschiedenen Stellen zur Definition und zum Nachweis von Termeigenschaften benutzt.
Die Semantik von Termen führt über die Interpretationsfunktion. Durch die Interpretation (über einer Rechenstruktur) wird eine semantische Äquivalenz auf Termen vorgegeben. Es lassen sich Regeln zur Umformung von Termen so gestalten, dass Terme stets in semantisch äquivalente Terme überführt werden können. Das Vorgehen führt zu den Termersetzungssystemen.

2.1 Termersetzungsregeln

Zunächst wird mit den Termersetzungsregeln der Kern eines Termersetzungssystems behandelt.

- Termersetzungsregel
  - Paar von Termen (t, r)
  - Regel-Schreibweise: $t \rightarrow r$
  - Terme t und r dabei
    1. von gleicher Sorte (zu einer gegebenen Signatur)
    2. mit freien Identifikatoren aus einer gegebenen Menge X von Identifikatoren
- Forderung: Alle Identifikatoren, die im Term r auftreten, müssen auch im Term t auftreten
- Instanz der Regel $t \rightarrow r$: $t[t_1/x_1, ..., t_n/x_n] \rightarrow r[t_1/x_1, ..., t_n/x_n]$
  - Ersetzen gewisser Identifikatoren $x_1, ..., x_n$ in t und r durch Terme $t_1, ..., t_n$ passender Sorten

Information 19: Termersetzungsregeln

Termersetzungsregeln sind Paare von Termen (t, r) mit den in Information 19 genannten Eigenschaften. Häufig wird die Forderung gestellt, dass die Menge der in t und r auftretenden Identifikatoren übereinstimmt. Es sei angemerkt, dass es auch Termersetzungssysteme gibt, bei denen eine Regel angewendet werden darf, auch wenn nicht sämtliche in der Regel auftretende Identifikatoren in beiden Termen vorhanden sind.

Ausgehend von einer Termersetzungsregel $t \rightarrow r$ gelangt man zu einer konkreten Ausprägung, genannt Instanz, indem festgelegt wird, welche Identifikatoren durch welche Terme in t und r zu ersetzen sind.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Betrachtete Regel $t \rightarrow r$
  - $pred(succ(x)) \rightarrow x$
- Gesucht sind mindestens zwei Regelinstanzen $t[t_1/x] \rightarrow r[t_1/x]$
  - _____
  - _____
  - _____
  - _____

Interaktion 20: Beispiel von Instanzen zu einer Regel

Das Vorgehen wird in Interaktion 20 an einem konkreten Beispiel verdeutlicht. Anhand der in diesem Beispiel betrachteten Regel $pred(succ(x)) \rightarrow x$ kann man sich die im Vergleich zu Textersetzungsregeln höhere Aussagekraft von Termersetzungsregeln klarmachen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- $c[t/x] \rightarrow c[r/x]$ heißt Anwendung einer Regel
  - $t \rightarrow r$ eine Instanz der Regel
  - c sei ein Term, in dem der Identifikator x frei vorkommt
  - t heißt Redex
  - das Auftreten von x in c heißt die Anwendungsstelle
- Beispiel:
  - Regelinstanz $t \rightarrow r$
    - $pred(succ(zero)) \rightarrow zero$
  - Anwendung der Regel auf den Term succ(succ(pred(succ(zero))))
    - $succ(succ(pred(succ(zero)))) \rightarrow succ(succ(zero))$
- Eine Menge von Termersetzungsregeln bildet einen Termersetzungsalgorithmus
  - analog zu den Textersetzungssystemen

Information 21: Regelanwendung und Termersetzungsalgorithmus

Regelinstanzen kommen bei der Anwendung einer Regel auf einen Term zum Einsatz, wie Information 21 zeigt. Durch die Anwendung einer Regel wird ein so genannter Termersetzungsschritt ausgeführt. Das beschriebene Vorgehen der Anwendung einer Regel, also die Ausführung eines Termersetzungsschritts, wird an einem Beispiel verdeutlicht.

Wie bei den Textersetzungssystemen führen auch die Termersetzungsregeln zum Algorithmenbegriff.

2.2 Termersetzungssystem und Termersetzungsalgorithmus

Bevor der Termersetzungsalgorithmus vertieft wird, muss zunächst das Termersetzungssystem formal eingeführt werden. Abschließend wird die Korrektheit von Termersetzungssystemen behandelt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ein Termersetzungssystem über einer Signatur Σ ist eine im Allgemeinen endliche Menge R von Termersetzungsregeln
- Berechnung ausgehend vom Term t_0
  - Folge von Termen $t_0, t_1, ..., t_n$, für die gilt:
    - $t_i \rightarrow t_{i+1}$ ist Anwendung einer Regel aus dem Termersetzungssystem R
  - t heißt terminal, wenn es keinen Term r gibt, so dass $t \rightarrow r$ gilt
- Normalform
  - Menge der terminalen Grundterme zu einem Termersetzungssystem
  - zu einem vorgegebenen Term t wird ein terminaler Term r als Normalform zugeordnet
    - liefert ein durch das Termersetzungssystem induziertes Normalformsystem

Information 22: Termersetzungssystem

Zu den in Information 22 beschriebenen Termersetzungssystemen gelangt man über die Termersetzungsregeln. Es wird mit einer endlichen Anzahl von solchen Regeln gearbeitet, wodurch die an Algorithmen gestellte Anforderung der endlichen Aufschreibung erfüllt wird.

Eine Folge von Regelanwendungen, die bei einem Ausgangsterm (hier t_0) beginnt, wird als Berechnung bezeichnet. Die Berechnung endet in natürlicher Weise, wenn ein Term erreicht wird, auf den keine der vorhandenen Regeln angewendet werden kann. Ein solcher Term (hier t_n) wird als terminal und die Berechnung als terminierend bezeichnet. Dieser Term heißt Ergebnis oder Ausgabe der Berechnung. Entsprechend heißt eine unendliche Folge t_i von Termen nicht terminierend.

Auf der Basis der terminierenden Berechnung bzw. der daraus resultierenden terminalen Terme lassen sich die so genannten Normalformen definieren. Als Normalform werden alle terminalen Grundterme zu dem Termersetzungssystem R bezeichnet. Man sagt auch, ein Term ist in Normalform bezüglich R.

Die Normalformen können dazu verwendet werden, einem beliebigen Term t eine Normalform r zuzuordnen, wobei r dann das Ergebnis einer Berechnung mit Eingabe t ist. Hierdurch wird aus dem Termersetzungssystem ein so genanntes Normalformsystem induziert.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ein Termersetzungssystem R mit einem Grundterm t als Eingabe definiert mittels der folgenden zwei Vorschriften einen Algorithmus:
  (1) Enthält R eine Ersetzungsregel mit Anwendung $t \rightarrow r$, dann wird der Algorithmus mit dem Term r statt t fortgesetzt.
  (2) Enthält R keine Ersetzungsregel mit Anwendung $t \rightarrow r$, so endet der Algorithmus mit r als Resultat.

- Beispiel: Termersetzungssystem zur Signatur der Rechenstruktur NAT
  - Sorte: nat
  - Funktionen: fct zero = nat fct succ = (nat) nat fct pred = (nat) nat
  - Normalformen: succ(...(succ(zero)...) für Terme der Sorte nat
  - Beispiel für einen Termersetzungsalgorithmus
    - Reduktionsregel für pred: pred(succ(x)) $\rightarrow$ x
    - Wirkung: Elimination des Funktionssymbols pred in Grundtermen
    - Berechnung: succ(pred(pred(succ(succ(pred(x)))))) $\rightarrow$

Interaktion 23: Termersetzungsalgorithmus

Die wiederholte Anwendung von Termersetzungsregeln definiert einen Algorithmus. Man gelangt zu diesem Termersetzungsalgorithmus über die in Interaktion 23 genannten zwei einfachen Vorschriften. Die erste Vorschrift behandelt dabei den Fall, dass es eine Ersetzungsregel gibt. Andernfalls gilt die zweite Vorschrift und der Algorithmus endet (bzw. terminiert). Der Anfangsterm t heißt entsprechend Eingabe und der Endeterm r heißt Resultat oder Ausgabe.

Bei dem angegebenen Beispiel handelt sich um ein Termersetzungssystem für die Signatur der Rechenstruktur NAT mit der Sorte nat und den drei Funktionen zero, succ und pred.
Die Normalformen bestehen offensichtlich aus einer beliebigen Anzahl von verschachtelten succ-Funktionen, die auf die 1-stellige Funktion zero angewendet werden.

Im beispielhaft angegebenen Termersetzungsalgorithmus wird durch die angegebene Regel pred(succ(x)) $\rightarrow$ x erreicht, dass in einem beliebigen (nicht undefinierten) Grundterm gewisse pred-Funktionen eliminiert werden.

Zu beachten ist, dass syntaktisch korrekte Terme wie pred(zero) mit der durch die natürlichen Zahlen gegebenen Interpretation einen undefinierten Wert ($\perp$) haben können. Solche Terme sind nicht in Normalform und es existiert auch kein semantisch äquivalenter Term in Normalform.

- Partielle Korrektheit
  - R: Termersetzungssystem; A: Rechenstruktur der Signatur
  - Eine Termersetzungsregel $t \rightarrow r$ über der Signatur heißt partiell korrekt bzgl. der Rechenstruktur A, falls für jede Belegung β in A gilt:

    $$I_\beta^A[t] = I_\beta^A[r]$$
  - R heißt partiell korrekt bezüglich der Rechenstruktur A, falls jede Regel partiell korrekt bezüglich A ist

- Totale Korrektheit
  - Ein partiell korrektes Termersetzungssystem R heißt total korrekt bezüglich der Rechenstruktur A, wenn

    (a) für Grundterme t mit $t^A \neq \perp$ keine nichtterminierenden Berechnungen existieren

    (b) für bezüglich R terminale, verschiedene Grundterme t_1, t_2 mit $t_1^A \neq \perp$ und $t_2^A \neq \perp$ stets gilt $t_1^A \neq t_2^A$

Information 24: Korrektheit von Termersetzungssystemen

Termersetzungsregeln entsprechen syntaktischen Ersetzungsregeln auf der Ebene der Terme. Beispielsweise sucht eine Regel wie pred(succ(x)) $\rightarrow$ x das Muster pred(succ(...)) in dem vorgegebenen Term und ersetzt dieses gemäß der Regel.

Die Bedeutung eines Terms innerhalb der Rechenstruktur A wird durch die Interpretationsfunktion I_β^A beschrieben, wobei β die Belegung der freien Identifikatoren im Term darstellt. Eine Termersetzungsregel $t \rightarrow r$ ist genau dann partiell korrekt bzgl. A, wenn $t = r$ eine in A gültige Aussage ist.

Umfassender als die partielle Korrektheit ist die vollständige Korrektheit, die zu der partiellen Korrektheit zwei weitere in Information 24 genannte Forderungen stellt.

Partielle und totale Korrektheit weisen hinsichtlich eines Aspekts eine grundsätzlich verschiedene Charakteristik auf: Während die partielle Korrektheit eine Eigenschaft der einzelnen Regeln ist (und damit auch für die einzelnen Regeln unabhängig gezeigt werden kann), ist die totale Korrektheit eine Eigenschaft des gesamte Termersetzungssystems.

3 MATHEMATISCHE LOGIK

Die mathematische Logik ist in der Informatik ein wichtiges Werkzeug, um Algorithmen zu beschreiben und zu verifizieren. Der so genannten Logikprogrammierung werden Systeme logischer Formeln als Algorithmenbeschreibungen aufgefasst. In diesem Kapitel wird der Aufbau aussagen- und prädikatenlogischer Formeln sowie der Ableitungsregeln (Inferenzregeln) behandelt [Br98].

- Die mathematische Logik dient in der Informatik als ein Werkzeug zur Beschreibung von Algorithmen
- Im Folgenden werden der Aufbau logischer Formeln und der logischen Regelsysteme behandelt
- Prinzip des „Tertium non datur“
  - es werden nur elementare Aussagen betrachtet, die entweder den Wert L oder den Wert O besitzen
  - ein dritter Wert ($\perp$) ist nicht zulässig
  - wird durch Beschränkung auf bestimmte Terme der Sorte bool sichergestellt
    - so genannte starke Terme
    - werden Formeln genannt
- Grundterme, die Formeln darstellen, heißen elementare Aussagen

Information 25: MATHEMATISCHE LOGIK - Einführung

Die Vermeidung von $\perp$ wird dadurch erreicht, dass nur solche Terme der Sorte bool zugelassen werden, für die aufgrund ihrer äußeren Gestalt sichergestellt ist, dass sie als Interpretation nur die Werte L und O besitzen, nicht aber den dritten Wert $\perp$. Das wird dadurch sichergestellt, dass in dem Term nur starke Funktionen verwendet werden. Eine Funktion heißt stark (*strong*), wenn sie für beliebige Argumente stets die Resultate L oder O liefert, also niemals das Resultat $\perp$ aufweist. Entsprechend spricht man von einem starken Term bzw. einer Formel, wenn ausschließlich starke Funktionen verwendet werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Rechenstruktur A, Sorte m; Terme t und r Grundterme der Sorte m
- Starke Gleichheit

$$(t = r)^A = \begin{cases} L \text{ falls } t^A = r^A \\ O \text{ falls } t^A \neq r^A \end{cases}$$

- Schwache Gleichheit

$$(t =^? r)^A = \begin{cases} L & \text{falls } t^A \neq \perp \text{ und } r^A \neq \perp \text{ und } t^A = r^A \\ O & \text{falls } t^A \neq \perp \text{ und } r^A \neq \perp \text{ und } t^A \neq r^A \\ \perp & \text{falls } t^A = \perp \text{ oder } r^A = \perp \end{cases}$$

Ist $t_1 = t_2$ eine Formel? _____

Ist $t_1 =^? t_2$ eine Formel? _____

Interaktion 26: Starke und schwache Gleichheit

Am Beispiel der starken und schwachen Gleichheit soll in Interaktion 26 die Frage verdeutlicht werden, ob ein Term der Sorte bool eine Formel ist oder nicht. Beide angegebenen Terme $t_1 = t_2$ und $t_1 =^? t_2$ sind Grundterme der Sorte bool.

- Gegenstand sind Formeln der Sorte bool, die
  - als Interpretation den Wert L besitzen
  - im Sinne der mathematischen Logik wahr sind
- Mathematische Logik behandelt das Formalisieren des Schlussfolgerns
  - Formeln: (zweiwertige) Terme der Sorte bool
  - elementare Aussage: Grundterme der Sorte bool
  - Ziel: Ableitung von Formeln aus einer Menge gegebener und als wahr angenommener Formeln
- In der Aussagenlogik sind die Formeln einfache Terme der Sorte bool
- Die Prädikatenlogik erweitert die Aussagenlogik, so dass auch auf andere Sorten bzw. Trägermengen Bezug genommen werden kann

Information 27: Inhalt und Ziel der mathematischen Logik

In der mathematischen Logik stehen solche Terme im Mittelpunkt der Untersuchung, deren Interpretation den Wert L ergibt. Diese Terme werden als wahr bezeichnet. Die Grundlage des Vorgehens in der mathematischen Logik bildet immer eine Rechenstruktur.

Häufig werden gewisse Formeln von vornherein als wahr vorausgesetzt. Diese Formeln werden auch als Hypothesen oder Axiome bezeichnet. Ausgehend von einer Hypothesen- oder Axiomenmenge wird durch die mathematische Logik ermittelt, welche weiteren Formeln hieraus ableitbar sind und entsprechend als wahr bewiesen werden können. Die abgeleiteten Formeln sind dann nicht Hypothesen oder Axiome, sondern so genannte Theoreme.

Das Mittel, auf dessen Basis die Ableitung („Formel-Beweis“) vorgenommen wird, ist ein Regelsystem, bestehend aus so genannten Ableitungs- oder Inferenzregeln. Die Regeln erzeugen also aus einer Menge von Axiomen ein Theorem, was als Beweis dieser abgeleiteten Formel anzusehen ist.

Das Vorgehen der Einführung eines Regelsystems liegt sowohl der Aussagenlogik als auch der Prädikatenlogik zugrunde. Die Prädikatenlogik kann dabei als Erweiterung der Aussagenlogik verstanden werden, da sie aufbauend auf den Termen der Sorte bool (Aussagenlogik) auf andere Sorten bzw. Trägermengen Bezug nimmt.
Die Logik ist ein Teilgebiet der Mathematik, das aufgrund der darin eingeführten Regelsysteme für die Informatik von hoher Relevanz ist.

3.1 Aussagenlogik

Zunächst werden die Aussagenlogik und der darin auftretende zentrale Ableitungsbegriff betrachtet.

- $H \vdash t$
  - heißt: Die Formel t ist aus der Formelmenge H ableitbar

- Ableitungsregeln der Aussagenlogik

(1)	$\vdash t \vee \neg t$	Tertium non datur
(2)	$\{t1, \neg t1 \vee t2\} \vdash t2$	Modus Ponens
	_____	Darstellung mit Implikationspfeil

(3) Ist t1 = t2 Anwendung einer Regel der semantischen Äquivalenzen, der Booleschen Algebra oder der Booleschen Terme, so gilt auch
$\{t1\} \vdash t2$ Anwendung Gleichheitsgesetze

Interaktion 28: Aussagenlogik

Es wird ein spezielles Ableitungssymbol $\vdash$ eingeführt. Wie gerade ausgeführt wurde, besteht das Ziel darin, aus einer Menge wahrer Formeln H gemäß gewisser Ableitungsregeln eine Formel t abzuleiten und damit als wahr zu beweisen. Die Axiomenmenge H kann dabei auch leer sein, was durch $\vdash t$ beschrieben ist. In der Aussagenlogik bestehen die in Interaktion 28 angegebenen drei Ableitungsregeln.

- Lemma
  Gilt $H \vdash t_a \Rightarrow t_b$, dann gilt auch $H \cup \{t_a\} \vdash t_b$

  Beweis

$H \vdash t_a \Rightarrow t_b$	Voraussetzung
$H \vdash \neg t_a \vee t_b$	Definition der Implikation
$\{t_a, \neg t_a \vee t_b\} \vdash t_b$	Modus Ponens
$H \cup \{t_a\} \vdash t_b$	w.z.b.w.

- Begriff der Ableitung formalisiert das Konzept des mathematischen Beweisens
  - Formales System oder Theorie
    = Axiomenmenge + Menge der Ableitungsregeln
  - Theoreme = Menge der ableitbaren Formeln

Information 29: Zusammenhang zwischen Implikation und Ableitbarkeit

Am Beispiel des in Information 29 angegebenen Lemmas, welches besagt, dass aus einer Implikation von t_b aus einem Term t_a (d.h. t_a impliziert t_b) auch dessen Ableitbarkeit folgt (also $H \cup \{t_a\} \vdash t_b$), soll das Vorgehen des Beweisens in der Aussagenlogik verdeutlicht werden.

- Hierbei wird im ersten Schritt von der Voraussetzung ausgegangen.

- Der zweite Beweisschritt nutzt die Definition der Implikation aus, die gleichwertig zum aussagenlogischen Term $\neg\ t_a \vee t_b$ ist. Unter der Annahme der Voraussetzung wurde hierdurch die Menge der aus der Hypothesenmenge H ableitbaren Formeln um den (starken) Term $t2 = \neg t_a \vee t_b$ ergänzt.
- Die letzten zwei Schritte stellen den eigentlichen Beweis des Lemmas dar, indem durch die Formulierung des zum Regelsystem gehörenden Modus Ponens die Herleitung der zu beweisenden Aussage möglich wird.

Der exemplarische Beweis verdeutlicht das grundsätzliche Vorgehen, dass die in einem früheren Schritt des Beweises abgeleiteten Aussagen zur Ableitung weiterer Aussagen verwendet werden. Das galt im obigen Fall für die Aussage $\neg t_a \vee t_b$, die in Schritt 2 erzeugt und in Schritt 4 dann verwendet wurde.

Der Begriff der Ableitung formalisiert das Konzept des mathematischen Beweises. Den Ausgangspunkt liefern die Axiome und Schlussregeln, die als formales System oder Theorie bezeichnet werden. Durch Anwendung der Schlussregeln lassen sich neue Formeln, die Theoreme, ableiten.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Eine Theorie heißt inkonsistent, falls jede Formel ableitbar ist

- Theorem: Ist in einer Theorie der Aussagenlogik false ableitbar, so ist jede Aussage t ableitbar, da gilt: {false} $\vdash$ t
  - Beweis

$\vdash t \vee \neg t$	Tertium non datur
$\vdash (t \vee t) \vee \neg t$	Idempotenz Disjunktion
$\vdash t \vee (t \vee \neg t)$	Assoziativität Disjunktion
$\vdash (t \vee \neg t) \vee t$	Kommutativität Disjunktion
$\vdash \neg \neg(t \vee \neg t) \vee t$	Involutionsgesetz
$\vdash \neg \neg$ true $\vee$ t	Gesetz für Boolesche Terme
$\vdash \neg$ false $\vee$ t	Gesetz für Boolesche Terme
$\vdash$ false $\Rightarrow$ t	Gesetz für Boolesche Terme
{false} $\vdash$ t	Zuvor bewiesenes Lemma mit

Interaktion 30: Inkonsistenz einer Theorie

In Interaktion 30 wird die Eigenschaft der Konsistenz bzw. Inkonsistenz einer Theorie (also eines Axiomensystems und den dazugehörigen Schlussregeln) behandelt. Falls in einer Theorie jede Formel ableitbar ist, so lässt sich zeigen, dass eine Theorie nicht konsistent sein kann. Formeln sind z.B. $\neg a \wedge a$ oder false. Falls jede Formel ableitbar ist, sind offensichtlich aussagenlogische Widersprüche erzeugbar.

Es gilt nicht nur, dass aus einer inkonsistenten Theorie false abgeleitet werden kann. Auch umgekehrt gilt: Für jede Theorie, aus der false abgeleitet werden kann (und somit false in der Axiomenmenge enthalten ist), kann bewiesen werden, dass diese Theorie inkonsistent ist.

Der Beweis hierzu ist in Interaktion 30 angegeben. Für den letzten Schritt des Beweises wird das zuvor bewiesene Lemma ausgenutzt, was im Rahmen der Interaktion zu zeigen ist.

Eine Theorie heißt korrekt, wenn alle ableitbaren Formeln für beliebige Belegungen der darin auftretenden freien Identifikatoren wahr sind. Eine Formel t ist dann wahr, wenn für alle Belegungen ß gilt, dass die Interpretationsfunktion $I_ß[t] = L$ ist.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Theorem

  Jeder aus der leeren Menge von Axiomen ableitbaren Aussage t mit freien Identifikatoren $x_1, \ldots, x_n$ ($n \in \mathbb{N}$) der Sorte bool wird für beliebige Belegungen von $x_1, \ldots, x_n$ durch L oder O der Wert L zugeordnet

  Beweis

  1. Aus der leeren Axiomenmenge ist unmittelbar nur $(t \vee \neg t)$ ableitbar
     - erfüllt die geforderte Eigenschaft
  2. Modus Ponens erlaubt aus wahren Aussagen nur die Ableitung wahrer Aussagen

- Ein formales System heißt vollständig, wenn für jede elementare Aussage t (jede Formel ohne freie Identifikatoren) die Formel t oder $(\neg t)$ ableitbar ist

Information 31: Korrektheit und Vollständigkeit

In Information 31 wird gezeigt, dass das formale System (die Theorie) mit der leeren Axiomenmenge und den drei eingeführten Ableitungsregeln der Aussagenlogik korrekt ist.

Eine noch umfassendere Anforderung als die Korrektheit wird an eine Theorie durch die Eigenschaft der Vollständigkeit gestellt. Diese Eigenschaft bedeutet, dass in der Theorie alle korrekten Formeln abgeleitet werden können.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Theorem
  - Wird eine aussagenlogische Formel t, die keine elementaren atomaren Aussagen enthält, mit freien Identifikatoren $x_1, \ldots, x_n$ der Sorte bool für jede Belegung von $x_1, \ldots, x_n$ mit Wahrheitswerten der Wert L zugeordnet, so ist t ableitbar

- Beweisidee
  - Es ist zu zeigen, dass einer Formel t genau dann der Wert L für alle Belegungen ß zugeordnet wird, wenn t auf true reduzierbar ist

Information 32: Vollständigkeit der Aussagenlogik

Ein Term t der Sorte bool heißt aussagenlogische Formel, wenn t nur aus

- den Termen true, false
- atomaren Aussagen (eine nicht weiter zerlegbare Aussage t_i, die wahr oder falsch sein kann)
- den logischen Operatoren
- Identifikatoren $x_1, \ldots, x_n$ der Sorte bool

aufgebaut ist.

Für jede Belegung von $x_1, ..., x_n$ mit Wahrheitswerten kann einer aussagenlogischen Formel, die keine elementaren atomaren Aussagen enthält, mittels der Interpretationsfunktion ein Wahrheitswert L oder O zugeordnet werden.

Die Aussagenlogik stellt eine wichtige Grundlage für Programmiersprachen dar, da in praktisch jeder Programmiersprache die Wahrheitswerte und die behandelten Abbildungen auftreten.

3.2 Prädikatenlogik

Der Aussagenlogik liegt eine Signatur zugrunde, in der nur Terme der Sorte bool auftreten. Die Prädikatenlogik lässt sich als eine Erweiterung der Aussagenlogik auffassen. Sie betrachtet Terme, die durch Anwendungen von Abbildungen auf Elemente gewisser Sorten mit Ergebnissen in der Menge der Wahrheitswerte entstehen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- n-stelliges Prädikat, n-stellige boolesche Funktion
  - eine Abbildung p: $M_1 \times ... \times M_n \rightarrow \mathcal{B}$ von Tupeln über gegebenen Mengen in die Menge der Wahrheitswerte
  - jede Anwendung des Prädikats p bezogen auf Elemente $a_i \in M_i$ liefert einen Wahrheitswert $p(a_1, ..., a_n)$
- Beispiele — Funktionsschreibweise
  - einstellig: x studiert Informatik _____
  - zweistellig: $\leq, \geq, =, \neq$ für Zahlen _____
    $\in$ für Mengen _____
- Nahe liegende Möglichkeiten, ein Prädikat p in eine elementare Aussage zu verwandeln:
  (1) für alle $x \in M$ gilt p(x)
  (2) für (mindestens) ein $x \in M$ gilt p(x)

Interaktion 33: Prädikatenlogik

Die Form von Abbildungen, die in der Prädikatenlogik betrachtet werden, bezeichnet man als Prädikate. Wie in Interaktion 33 dargestellt ist, kommen mit den Prädikaten neue nicht-boolesche Sorten bzw. Mengen zur Rechenstruktur hinzu. Durch das Prädikat werden diese auf einen booleschen Wahrheitswert abgebildet.

Je nach Anzahl n der als Argumente in das Prädikat eingehenden Argumente spricht man von einem n-stelligen Prädikat. Ein Beispiel eines Prädikats mit einem Eingangsargument, also ein einstelliges Prädikat, ist x studiert Informatik. Entsprechend sind die verschiedenen Gleichheitsoperatoren oder der Enthaltenseinsoperator Beispiele für zweistellige Prädikate.

Das Ergebnis der so konstruierten Prädikate ist von den jeweiligen Werten abhängig, die die Eingangsagrumente besitzen. Offensichtlich lassen sich durch die zwei Konstrukte für alle und es gibt Aussagen treffen, die den gesamten Wertebereich der Eingangsargumente einbeziehen. Es sind allgemeingültige und erfüllbare Prädikate wie folgt definiert:

- Ist eine Aussage der Form für alle $x \in M$ gilt p wahr, so gilt: Prädikat p ist allgemeingültig.

• Ist eine Aussage der Form für (mindestens) ein x∈M gilt p wahr, so gilt: Prädikat p ist erfüllbar.

Nachfolgend werden Aussagen der obigen Form formal als Terme der Prädikatenlogik eingeführt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- $W_{\Sigma}^{bool}(X)$ mit $\Sigma = (S, F)$ beschreibt die Menge der booleschen Terme
  - Σ ist eine Signatur, die u.a. die Sorte bool und die booleschen Operatoren $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$ enthält
  - Operationssymbole aus F
  - freie Identifikatoren aus der Familie von Mengen von Identifikatoren X

- Prädikatenlogische Formeln
  - $\forall$ m x: t (Allquantor)
    - für alle Elemente x ($\neq \perp$) der Sorte m gilt die Aussage t
  - $\exists$ m x: t (Existenzquantor)
    - es existiert ein Element x ($\neq \perp$) der Sorte m, für das die Aussage t gilt

- Bei Formeln der obigen Form spricht man von Quantifizierung

Information 34: Bildung prädikatenlogischer Formeln

Die Informatik ist insbesondere auch an der (Term-) Algebra interessiert, die sich aus den über einer Algebra gebildeten Termen ergibt (siehe auch die Ausführungen in der Kurseinheit ALGEBRAISCHE STRUKTUREN UND FORMALE SYSTEME). $W_{\Sigma}^{bool}(X)$ bezeichnet die Menge der booleschen Terme, auf deren Grundlage die Termalgebra der Prädikatenlogik gebildet wird. Die Termalgebra beinhaltet die korrekt gebildeten Formeln (*well formed formula* wff).

Aufbauend auf einer Formel t, die Identifikatoren der Sorte m der Trägermenge M enthält, gelangt man durch die Einführung von zwei neuen Symbolen zu den prädikatenlogischen Formeln. Anstelle der zuvor angegebenen Begriffe für alle bzw. es gibt sind die Schreibweisen $\forall x \in M$: t bzw. $\exists x \in M$: t üblich. Die Symbole heißen Quantoren, deren Anwendung wird als Quantifizierung bezeichnet.

- In Quantifizierungen treten Identifikatoren nicht frei, sondern durch den All- bzw. Existenzquantor gebunden auf
  - in einer Substitution eines gebundenen Identifikators durch einen Term wird dieser nicht ersetzt
  - ausgedrückt durch folgende Regeln
    $(\forall\ m\ x: t)\ [t'/x] = \forall\ m\ x: t$
    $(\exists\ m\ x: t)\ [t'/x] = \exists\ m\ x: t$

- Beispiele prädikatenlogischer Ausdrücke mit Quantoren

	wahr	falsch
• $\forall$ bool x: x $\vee \neg$x	☐	☐
• $\exists$ nat m: $\forall$ nat n: n < m	☐	☐
• $\forall$ nat n: $\exists$ nat m: n < m	☐	☐

- Gesetze zu den Quantoren
  1. $(\forall\ m\ x: t) = (\neg\ \exists\ m\ x: \neg\ t)$
  2. $(\exists\ m\ x: t) = (\neg\ \forall\ m\ x: \neg\ t)$

Interaktion 35: Eigenschaften von Quantifizierungen

Die Semantik des in Interaktion 35 eingeführten Begriffs "gebunden" wird im Zusammenhang mit der Substitution deutlich.

An den Beispielen wird deutlich, dass man wahre und falsche Aussagen mittels Quantoren bilden kann. Am zweiten und dritten Beispiel lässt sich leicht klarmachen, dass sich durch Änderung der Reihenfolge der Quantoren der Wahrheitswert der Aussage ändern kann.

Durch die zwei in Interaktion 35 angegebenen Quantorengesetze wird deutlich, dass sich der Allquantor in Form einer Negation durch den Existenzquantor ausdrücken lässt und umgekehrt. Die Gesetze sind von ihrem Sinn her leicht nachvollziehbar. So besagt Gesetz (1), dass man kein x finden kann, für das $\neg$t gilt, falls für alle x der Term t gilt. Das lässt sich graphisch so veranschaulichen, dass die Menge der Identifikatoren, für die $\neg$t gilt, leer ist.

Sei M die Trägermenge zur Sorte m und $M \neq \varnothing$

(1) Gilt $(\Sigma, H) \vdash t$ und ist x nicht frei in den Formeln in H und nicht Funktionssymbol in Σ, so gilt:
$(\Sigma, H) \vdash \forall$ m x: t

(2) Sei t1 ein Term der Sorte m, wobei $t1 \neq \perp$; gilt $(\Sigma, H) \vdash \forall$ m x: t, so gilt:
$(\Sigma, H) \vdash t[t1/x]$

(3) Gilt $(\Sigma, H) \vdash t[t1/x]$ für einen Term t1 der Sorte m, wobei $t1 \neq \perp$, so gilt:
$(\Sigma, H) \vdash \exists$ m x: t

(4) Gegeben die Signatur $\Sigma = (S, F)$ mit fct x = m und $\Sigma' = (S, F \setminus \{x\})$;
gilt $(\Sigma', H) \vdash \exists$ m x: t, dann gilt, falls x nicht frei in H:
$(\Sigma, H) \vdash$ t

Information 36: Ableitungsregeln der Prädikatenlogik

Ableitungsregeln wurden bereis im Zusammenhang mit der Aussagenlogik eingeführt. Wie Information 36 zeigt, bestehen insgesamt vier Regeln zur Ableitung von prädikatenlogischen Formeln über der Signatur $\Sigma = (S, F)$.

Auf die in prädikatenlogischen Termen möglichen Umformungen der Umbenennung und der Substitution wird in Information 37 eingegangen.

- Umbenennung von gebundenen Identifikatoren
  - $(\forall$ m x: t) = $\forall$ m y: (t[y/x]) falls y nicht frei in t vorkommt
  - $(\exists$ m x: t) = $\exists$ m y: (t[y/x]) falls y nicht frei in t vorkommt

- Substitutionsregel für quantifizierte prädikatenlogische Terme
  (1) $(\forall$ m x: t)[t´/x] = $\forall$ m x: t
  (2) $(\forall$ m x: t)[t´/y] = $\forall$ m x: (t[t´/y])
  - falls x und y verschiedene Identifikatoren sind und x nicht frei in t´ vorkommt

- Wertezuordnung zu einer prädikatenlogischen Formel (Semantik)

$$I_ß[\forall m\ x:\ t] = \begin{cases} L & \text{falls für alle } a \in M \text{ (mit } a \neq \perp\text{): } I_{ß[a/x]}[m\ x:\ t] = L \\ 0 & \text{sonst} \end{cases}$$

Information 37: Umbenennung und Substitution

Eine Umbenennung ist möglich, wenn die Forderung der Gebundenheit des umzubennenden Identifikators erfüllt ist. Wie in Information 37 ausgeführt ist, gilt das für den Allquantor und den Existenzquantor in gleicher Weise.

Bei der Substitution eines Identifikators ist zu unterscheiden, ob dieser im prädikatenlogischen Term gebunden oder frei ist. Die Beschreibung der Regel erfolgt in Form der zwei in Information 37 angegebenen Gleichungen.

Die Interpretationsfunktion wird dazu genutzt, die Semantik der Quantoren festzulegen, wie am Beispiel des Allquantors in Information 37 gezeigt wird. Die Interpretation einer prädikatenlogischen Formel mit dem Existenzquantor (∃ m x: t) erfolgt analog, wobei für alle gegen für ein zu ersetzen ist.

Das in prädikatenlogischen Termen verwendete Konzept der Bindung von Identifikatoren und die auftretenden Umbenennungsregeln finden sich in analoger Form auch in Programmiersprachen wieder.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Eine Struktur A ist ein Paar (U_A, I_A) mit einer nichtleeren Menge U_A und einer Abbildung I_A, die
  - jedem k-stelligen Prädikatensymbol P ein k-stelliges Prädikat über U_A zuordnet
  - jedem k-stelligen Funktionssymbol f eine k-stellige Funktion auf U_A zuordnet
  - jeder Variablen x ein Element aus U_A zuordnet
- Eine Struktur A ist ein Modell für eine Formel F, falls F in der Struktur A wahr ist
- Eine Formel F ist erfüllbar, wenn sie mindestens ein Modell besitzt
- Beispiel: $U_A = \mathbb{N}$ $I_A(P) = \{(m, n) \mid m, n \in \mathbb{N}, m < n\}$
  - Ist die Formel $F = \exists x \exists y \exists z (P(x, y) \wedge P(z, y) \wedge P(x, z) \wedge \neg P(z,x))$ ein Modell?

Interaktion 38: Struktur und Modell

Abschließend soll in Interaktion 38 die Belegung, die der Interpretationsvorschrift I zugrunde liegt, näher beleuchtet werden, was zu dem Begriff der Struktur führt. Eine Struktur besteht aus zwei Elementen, einer Interpretation I sowie einer Menge U. Um zu verdeutlichen, dass I und U zu der Struktur A gehören, erscheint das A als Index (also I_A und U_A). U als Bezeichnung der Menge steht dabei für den Begriff des Universums.

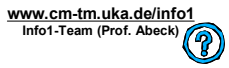
Wie die Definition der Struktur verdeutlicht, ordnet die Interpretation Prädikatensymbole, Funktionssymbole und Variablenbezeichner bestimmten Prädikaten, Funktionen und Variablen des Universums zu. Die Menge U_A muss selbstverständlich die geforderten Prädikate und Funktionen mit der geforderten Stelligkeit enthalten.

Die Prädikate, Funktionen und Variablen treten sich in entsprechenden prädikatenlogischen Formeln auf. Auf der Basis einer Struktur-Definition, wie sie oben gegeben ist, lassen sich mittels der Interpretationsvorschrift die in einer bestimmten Formel F enthaltenen Prädikate, Funktionen und Variablen des Universums einsetzen und der Wert der Formel bestimmen.

Wenn es möglich ist, eine Struktur anzugeben, die ein Modell für eine Formel F ist, wird F erfüllbar genannt. Entsprechend ist F nicht erfüllbar, falls gezeigt werden kann, dass es kein Modell geben kann. Die Begriffe "Modell" und "Struktur" werden in Interaktion 38 anhand eines einfachen Beispiels verdeutlicht.

4 FUNKTIONALE PROGRAMMIERKONZEPTE

Funktionale Programmierkonzepte – die auch synonym als applikative Programmierkonzepte bezeichnet werden – stehen in einem engen Zusammenhang mit den zuvor beschriebenen Termersetzungssystemen. Wie in Interaktion 39 ausgeführt ist, sind funktionale Programme die Terme. Die Ausführung des Programms stellt eine durch ein Rechensystem realisierbare Termersetzung dar [Br98].



- Ein Programm lässt sich als ein Term auffassen
  - kann auf einem Rechensystem ausgeführt werden, falls ein allgemeiner Auswertungsalgorithmus (“Interpretierer”) verfügbar ist
  - im Term treten Funktionen auf
- Das zentrale in einem funktionalen Programm auftretende Sprachelement ist die Funktionsanwendung
- Beispiel: Fakultätsfunktion !
  - mathematische Spezifikation

$.!: \mathbb{N} \rightarrow \mathbb{N}$ $0! = 1,$
$(n)! = (n-1)! \cdot n$

$n \in \mathbb{N},\ n! = \begin{cases} \_\_\_\_\_ \\ \_\_\_\_\_ \end{cases}$

Interaktion 39: FUNKTIONALE PROGRAMMIERKONZEPTE – Funktionales Programm

Der allgemeine Ausführungsalgorithmus zu einer Programmiersprache entspricht den Reduktionssystemen bei den Termersetzungssystemen. Sofern diese Form des Reduktionssystems, also der Interpretierer, auf einem Rechensystem ausgeführt werden kann, sind die Algorithmen auf dem Rechensystem ablauffähig.

Das beherrschende Sprachelement der funktionalen Programmiersprachen ist die Funktionsanwendung, die auch als Funktionsapplikation bezeichnet wird. Dieses Sprachelement tritt in jeder höheren Programmiersprache auf. Funktionale Sprachen beschränken sich auf dieses Sprachelement und verzichten auf ablauforientierte Strukturen, wie z.B. Schleifen. Bei der funktionalen Programmierung erfolgt eine Reduktion auf die zur Programmierung notwendigsten Sprachelemente.

Als Beispiel wird in Interaktion 39 die Fakultätsfunktion eingeführt, die mathematisch durch eine sich auf den Wert von n beziehende Fallunterscheidung beschrieben ist. Ein funktionales Programm, das die Fakultätsfunktion berechnet, könnte z.B. als Java-Programm formuliert werden, wie im Folgenden gezeigt wird.

- Zahlreiche gegebene Funktionssymbole und Operationen werden vorausgesetzt
  - z.B. Addition (+), Vergleich (==)
- Die Fallunterscheidung (if-then-else) ist ein Element der funktionalen Sprache
  - wird nicht als eine (nicht-strikte) Funktion aufgefasst, sondern als fester Bestandteil der Sprache
- Fakultätsfunktion als funktionales Programm (Java-Syntax)

```
static long fact (int n) {
  if (n == 0) return 1;
  else return fact(n - 1) * n;
}
```

- Genutzte Funktionssymbole

_____

- Genutzte Sprachelemente

_____

_____

Interaktion 40: Elemente eines funktionalen Programms

Die als gegeben vorausgesetzten Operationen bilden die so genannte algorithmische Basis. Sie werden als elementare oder primitive Operationen bzw. primitive Funktionen bezeichnet. Hierzu zählt neben den arithmetischen Operationen auch die Fallunterscheidung, die als Element der funktionalen Sprache angesehen wird.

Am Beispiel des in Interaktion 40 angegebenen funktionalen Programms, das die zuvor eingeführte mathematische Beschreibung der Fakultätsfunktion umsetzt, lassen sich die genutzten Sprachelemente identifizieren.

Bei diesem Beispiel eines funktionalen Programms tritt eine Rekursion auf. Auf die Rekursion wird im nächsten Kapitel näher eingegangen, nachdem die elementaren Konzepte von funktionalen Programmen eingeführt wurden.

- Ein funktionales Programm besteht aus der Definition von Funktionen und deren Applikation innerhalb von Termen

- Beinhaltet der Term nur primitive Operationen, so heißt er primitiver Term, andernfalls (nichtprimitiver) Ausdruck

  - Welche Art von Term liegt bei dem zuvor behandelten funktionalen Programm vor, das die Fakultät berechnet?

    _____

  - Was muss gegeben sein, damit sich die Fakultätsfunktion funktional in Form eines primitiven Terms berechnen lässt?

    _____

Interaktion 41: Funktion und Funktionsanwendung

Da das zentrale Sprachmittel der funktionalen Programmiersprache die Funktionsanwendung ist, besteht ein funktionales Programm aus der Definition von Funktionen und deren Anwendung in Termen. Die Sprachelemente führen unmittelbar zu einer Aufteilung der in einem funktionalen Programm auftretenden Terme. Es bestehen

- primitive Terme, die nur primitive Funktionen beinhalten und
- nichtprimitive Ausdrücke, die zuvor definierte Funktionen anwenden.

Am Beispiel der Fakultätsfunktion sind in Interaktion 41 die beiden Arten von Terme aufzuzeigen.

4.1 Syntax und Semantik eines funktionalen Programmen

Nachfolgend werden der Aufbau (Syntax) eines funktionalen Programms und dessen Bedeutung (Semantik) behandelt.

- Die formale Sprache der Ausdrücke (Expressions Expr) in Backus Naur Form (BNF)

Expr	=	id | ⊥ | FunctionApplication | ConditionalExpression | (Expr) | MonadOp Expr | Expr DyadOp Expr | Block.
MonadOp	=	"-" | "¬".
DyadOp	=	"+" | "-" | "*" | "/" | "<" | "==" | ">" | ">=" | "∧" | "∨".

- Forderung weiterer syntaktischer Restriktionen in Form von Kontextbedingungen
  - z.B. jedem Ausdruck kann genau eine Sorte zugeordnet werden

Information 42: Syntax von Ausdrücken

Ein Ausdruck kann sein

- ein Identifikator, wobei auch das die Undefiniertheit angebende *Bottom*-Element ⊥ als spezieller Identifikator zugelassen wird,
- eine Funktionsanwendung oder ein bedingter Ausdruck (if),
- ein geklammerter oder ein in Infix- oder Präfix-Notation gebildeter Ausdruck,
- eine zur Einführung von lokalen Deklarationen dienende syntaktische Einheit Block, die (wie auch verschiedene andere Nichtterminale) später im Detail eingeführt werden.

Danach sind die monadischen (1-stelligen) bzw. dyadischen (2-stelligen) primitiven Operationen angegeben, die im Ausdruck auftreten dürfen. Dabei wird zwischen arithmetischen Operationen und booleschen Operationen unterschieden.

Die in Information 42 in Backus Naur Form (BNF, siehe auch Kurseinheit PROGRAMMIERGRUNDLAGEN) formulierten Regeln beschreiben den syntaktischen Aufbau von Ausdrücken. Es gibt noch weitere den syntaktischen Aufbau einschränkende Bedingungen (so genannte Kontextbedingungen), die neben den BNF-Regeln formuliert werden. Eine solche Kontextbedingung ist beispielsweise, dass jedem Ausdruck eine Sorte zugeordnet werden können muss. Diese Sorte wird durch das Ergebnis des Ausdrucks bestimmt und ist abhängig von der Zuordnung von Sorten zu den auftretenden Identifikatoren.

Programmiersprachen, für die diese Kontextbedingung gilt, heißen stark typisierte Sprachen. Eine starke Typisierung bietet einige Vorteile, wie z.B. die automatische Erkennung von Programmierfehlern aufgrund von Typkonflikten oder eine bessere Strukturierung und Lesbarkeit von Programmen.

Sehr viel schwieriger als die Syntax ist die Bedeutung der syntaktischen Beschreibung, also die Semantik des funktionalen Programms, festzulegen.

- Operationelle Semantik
  - Termersetzungssemantik
  - wird durch einen Termersetzungsalgorithmus dargestellt
  - Ausdrücke werden unter gewissen Nebenbedingungen in eine Normalform gebracht
  - nicht-wohldefinierte Terme (d.h. Terme t, für die $I(t) = \perp$ gilt) besitzen keine terminierende Berechnung

- Funktionale Semantik
  - Definition einer Interpretationsfunktion I
  - bei der Festlegung bildet eine Rechenstruktur die Grundlage
  - Ausdrücke der Programmiersprache werden auf mathematische Elemente (ihre Werte) abgebildet
  - Datenelemente für die semantische Interpretation von Ausdrücken
    - $D := \{a \in s^A : s \in S\}$
  - Menge der Funktionen, die in den Ausdrücken verwendet werden:
    - $FCT := \{f : s_1^A \times ... \times s_n^A \rightarrow s_{n+1}^A : n \in \mathbb{N} \wedge f \text{ strikt} \wedge \forall i, 1 \leq i \leq n+1 : s_i \in S\}$

Information 43: Semantik von Ausdrücken

Es werden grundsätzlich zwei verschiedene Semantik-Formen, die funktionale und die operationelle Semantik unterschieden. Funktionalen Programmiersprachen kann eine funktionale, aber auch eine operationelle Semantik zugeordnet werden. Ganz analog kann den imperativen Programmiersprachen neben einer operationellen auch eine funktionale Semantik zugewiesen werden.

Wird die operationelle Semantik auf funktionale Programme angewendet, so resultiert hieraus eine Termersetzungssemantik. Das Ziel der Termersetzung, die durch einen entsprechenden Algorithmus erfolgt, ist dabei die Erzielung einer gewissen Normalform.

Es ist zu beachten, dass Terme auch nicht wohldefiniert sein können: Ein nicht-wohldefinierter Term t ist dadurch charakterisiert, dass für die Interpretationsfunktion $I(t) = \perp$ gilt.
Die hier zur Charakterisierung eines nicht-wohldefinierten Terms benutzte Interpretationsfunktion I führt direkt zur funktionalen Semantik. Diese stützt sich auf Abbildungen, also Funktionen von gewissen mathematischen Elementen ab, die sich in Form der Menge D fassen lassen. Die Grundlage bildet eine Rechenstruktur.

4.2 Termersetzungskonzept

Bezüglich der Anwendungsmöglichkeit der Termersetzungsregeln wird bei der operationellen Semantik für Programmiersprachen von den Termersetzungssystemen abgewichen, da Termersetzungsregeln nicht an einer beliebigen Stelle eines Terms angewendet werden dürfen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Für Termersetzungssysteme gilt, dass Termersetzungsregeln in Termersetzungsalgorithmen an beliebigen Stellen in einem Term angewendet werden dürfen
  - gilt nicht für die operationelle Semantik von Programmiersprachen

- Effizienzsteigerung
  - für die Berechnung des Resultats nicht benötigte Auswertungen können unterdrückt werden

```
if (true) { // Anweisungsfolge 1}
else {// Anweisungsfolge 2}
```

Effizienzsteigerung, weil

_____

- Terminierungseigenschaft
  - nichtterminierende, aber nicht benötigte Teilausdrücke gefährden die Terminierung des Gesamtausdrucks nicht

Terminierungsgefährdung, falls

_____

Interaktion 44: Termersetzungskonzept

Das eingeschränkte Termersetzungskonzept weist die Vorteile der Effizienz und der Terminierungseigenschaft auf, wie anhand des in Interaktion 44 angegebenen Beispiels zu verdeutlichen ist. Die Forderung nach einer Einschränkung der Anwendung von Termersetzungsregeln führt zu den bedingten Termersetzungsregeln.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Aussehen von bedingten Termersetzungsregeln
  - $t_1 \rightarrow r_1 \wedge ... \wedge t_n \rightarrow r_n \Rightarrow t_{n+1} \rightarrow r_{n+1}$

- Beispiel: Funktion cor (conditional or, bedingtes Oder)
  - fct cor = (bool, bool) bool
  - Auswertungsregeln:
    1. $x \rightarrow z \Rightarrow cor(x, y) \rightarrow cor(z, y)$,
    2. $cor(true, y) \rightarrow true$,
    3. $cor(false, y) \rightarrow y$.
  - Ergebnis: $cor(t_1, t_2)$ terminiert auch dann sicher,
    - wenn Term t_1 terminiert und true liefert
    - selbst für den Fall, dass Term t_2 nicht terminiert

Wertetabelle zu cor

cor	l(t2) L	O	⊥
l(t1) L			
O			
⊥			

Interaktion 45: Bedingte Termersetzung

Eine bedingte Termersetzungsregel bedeutet: Die Ersetzung des Term t_{n+1} durch r_{n+1} ist nur unter der Bedingung zulässig, dass zuvor die Terme t_i durch r_i ($i = 1, .., n$) ersetzt wurden.

Die bedingten Termersetzungsregeln werden in Interaktion 45 am Beispiel des bedingten Oder, dem *conditional or* (cor) verdeutlicht.

Anhand der in Interaktion 45 aufzustellenden Wertetabelle wird ersichtlich, dass die cor-Funktion an genau einer Stelle einen zur (normalen) or-Funktion unterschiedlichen Wertverlauf aufweist.

4.3 Identifikatoren

Identifikatoren traten bereits im Zusammenhang mit dem Aufbau von Termen auf. In Programmiersprachen haben Identifikatoren eine große Bedeutung (siehe Information 46).

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Identifikatoren werden in Programmiersprachen für unterschiedliche Zwecke genutzt, wie z.B.
  - zur Bezeichnung von Zwischenergebnissen
  - als Parameter in Ausdrücken
  - zur Bezeichnung von Funktionen
- Syntaktischer Aufbau von Identifikatoren
  - Beispiel: erstes Zeichen muss ein lateinischer Buchstabe sein
    - hierdurch eine Unterscheidung von Zahlen möglich
- In funktionalen Programmiersprachen stehen Identifikatoren für
  - Elemente aus der Menge D (Elemente der Trägermengen)
  - Elemente aus der Menge FCT (n-stellige Funktionen)

Information 46: Identifikatoren in Programmiersprachen

An den Aufbau von Identifikatoren werden in Programmiersprachen üblicherweise gewisse syntaktische Anforderungen gestellt. Durch die Vorgabe, dass Identifikatoren mit einem Buchstaben beginnen müssen, lassen sie sich von Zahlen unterscheiden. Diese Anforderung lässt sich leicht durch eine entsprechende BNF-Regel formulieren.

Schwieriger zu behandeln ist die Frage der Semantik von Identifikatoren in funktionalen Programmen. Zum einen kann sich hinter einem Identifikator ein Datenelement, also ein Element aus den Trägermengen verbergen (diese Menge wurde mit D bezeichnet) oder es handelt sich um eine Funktion, bezeichnet mit der Menge FCT. Die Mengen D und FCT bilden gemeinsam die Menge H der so genannten semantischen Elemente, d.h. $H =_{def} D \cup FCT$.

- Ausdrücken mit freien Identifikatoren wird eine Bedeutung dadurch zugeordnet, dass diese Identifikatoren belegt werden
  - ENV := {β: ID $\rightarrow$ H}
  - I:<exp> $\rightarrow$ (ENV $\rightarrow$ H)

- Interpretation einzelner Identifikatoren
  - $I_\beta[x] = \beta(x)$
  - Einführung einer speziellen Abbildung Ω, die allen Identifikatoren das Element $\perp$ zuordnet

$$\Omega : ID \rightarrow H \text{ mit } \Omega(x) = \perp$$

- Konstanten sind Zeichen und Zeichenfolgen mit fester, von der Belegung unabhängiger Interpretation
  - Deutung als 0-stellige Funktionen der Rechenstruktur möglich

Information 47: Zuordnung von Bedeutungen zu Identifikatoren

Wie in Information 47 näher ausgeführt ist, wird die Menge H der semantischen Elemente benötigt, um den Identifikatoren eine Bedeutung, also eine Semantik zuzuordnen.

Eine Abbildung β weist jeder syntaktischen Beschreibung eines Identifikators (zusammengefasst in der Menge ID) ein semantisches Element aus H zu. Die Abbildung β heißt eine Belegung.

Die Menge aller Belegungen wird mit ENV (*Environment*, Umgebung) bezeichnet. Mittels der Umgebung ENV lässt sich auch einem beliebigen Ausdruck, in dem Identifikatoren aus der Umgebung vorkommen, ein semantischer Wert aus H zuordnen. Die Abbildung, die das leistet, ist die Interpretationsvorschrift I.

Die Interpretationsvorschrift I lässt sich auf einen einzelnen Identifikator x anwenden (dieser bildet ebenfalls einen Ausdruck). Gemäß der Definition der Umgebung ENV ist die Interpretation eines Identifikators die Belegungsabbildung

Das von $\beta(x)$ einem Identifikator zugewiesene semantische Element kann auch das undefiniert-Element $\perp$ sein, weil dieses zu H gehört. Es ist somit eine korrekte Belegung, jedem Identifikator $\perp$ (Undefiniert-Element) zuzuordnen. Diese Belegung wird speziell gekennzeichnet und als Ω-Abbildung bezeichnet.

Konstanten haben üblicherweise den gleichen syntaktischen Aufbau wie Identifikatoren, aber eine unterschiedliche Semantik. Der semantische Unterschied besteht darin, dass Konstanten unabhängig von der Belegung ein semantisches Element darstellen.

Mit den Identifikatoren und Konstanten sowie deren Syntax und Semantik wurde bereits ein Sprachelement einer funktionalen Programmiersprache vorgestellt. Dieses Sprachelement wird im Folgenden durch weitere ergänzt. Es wird dabei von der Annahme ausgegangen, dass die Rechenstrukturen BOOL und INT vorhanden sind. Das bedeutet konkret, dass die in den entsprechenden Signaturen vorhandenen Sorten und Funktionen in den Programmen verwendet werden.

4.4 Bedingte Ausdrücke

Die im ersten Kapitel dieser Kurseinheit ausführlich beschriebenen Rechenstruktur BOOL und der darin auftretenden Sorte bool wird in den in Information 48 beschriebenen bedingten Ausdrücke und des darin verwendeten booleschen Ausdrucks B genutzt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ein bedingter Ausdruck der Sorte s hat den folgenden Aufbau

  if (B) E else E´

  (1) B: Ausdruck der Sorte bool

  (2) E, E´: beliebige Ausdrücke gleicher Sorte s

- Syntax:

ConditionalExpression = "if" "(" Expr ")" Expr ["else" Expr].

- Semantik:

$$I_\beta[\text{if (B) E else E´}] = \begin{cases} I_\beta[E] & \text{falls } I_\beta[B] = L \\ I_\beta[E´] & \text{falls } I_\beta[B] = O \\ \bot & \text{sonst} \end{cases}$$

Information 48: Bedingte Ausdrücke

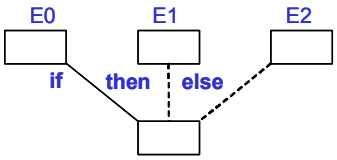
E und E´ heißen Zweige. Die beiden Anforderungen an die Sorte von B, E und E´ sind syntaktische Nebenbedingungen. Beide Nebenbedingungen machen eine Vorgabe an die erwarteten Typen bzw. Sorten. Zu beachten ist, dass eine stark typisierte Sprache zugrunde gelegt wird.

Die semantische Interpretation I_β besagt, dass der bedingte Ausdruck einer nichtstrikten Abbildung entspricht: Auch wenn die Interpretation eines Ausdruck in einem der Zweige einen undefinierten Wert ($\bot$) liefert, kann der Wert des bedingten Ausdrucks verschieden von $\bot$ sein.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Beschreibung der Semantik durch folgende Axiome
  1. if (true) E else E´ = E
  2. if (false) E else E´ = E´
  3. if (B) E else E´ = if ($\neg$B) E´else E
  4. if ($\bot$) E else E´ = $\bot$

- Beschreibung durch Formulare



Das Formular ist so auszufüllen, dass obiges Axiom 1 beschrieben wird

Interaktion 49: Axiome und Formulare zu bedingten Ausdrücken

Die in Interaktion 49 angegebenen Axiome in der Gestalt von Gleichungen beschreiben die Semantik bedingter Ausdrücke. Das Gleichungssystem gestattet, zu jeder denkbaren

Wertebelegung einen Resultatwert abzuleiten. In diesem Zusammenhang wird von einer Wertverlaufssemantik gesprochen. Diese Art von Semantik kann z.B. auch zur Festlegung der Bedeutung von aussagenlogischen Termen genutzt werden.

Die Axiome stellen somit eine bestimmte Art der Semantikbeschreibung dar. Als eine mögliche Syntaxform wurden bereits die Formulare eingeführt. Für bedingte Ausdrücke hat das (Berechnungs-) Formular das in Interaktion 49 angegebene Aussehen. Durch die gestrichelten Linien wird ausgedrückt, dass die Ausdrücke (hier E1 und E2) nicht immer ausgewertet werden müssen. Das Formular zum bedingten Ausdruck ist demgemäß in folgender Weise zu deuten:

- Zuerst ist E0 auszuwerten.
- Dann erfolgt die Auswertung entweder von E1 oder von E2 in Abhängigkeit des ermittelten Werts für E0,

Aus diesem Auswertungsvorgehens resultiert die Nicht-Striktheit.

4.5 Funktionsanwendung und Funktionsabstraktion

Das Sprachelement der Funktionsanwendung und die damit eng verbundene Funktionsabstraktion haben eine zentrale Bedeutung in den funktionalen Sprachen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- $F(E_1, ..., E_n)$ heißt Funktionsanwendung
  - äquivalente Bezeichnungen sind Funktionsapplikation oder Funktionsaufruf
- Nebenbedingungen
  - die durch F bezeichnete Funktion muss n-stellig sein
  - die Sorten der $E_1, ..., E_n$ müssen der Funktionsfunktionalität entsprechen
- $f(a_1, ..., a_n)$ ist der Wert der Funktionsanwendung
  - f ist die durch F bezeichnete Funktion
  - $a_1, ..., a_n$ sind die Werte der Ausdrücke
- Syntax der Funktionsanwendung
  - als BNF
  - als Syntaxdiagramm

```
FunctionApplication= Function "(" [Expr {"," Expr}] ")".
```

Interaktion 50: Funktionsanwendung

Durch die Funktionsapplikation bzw. -anwendung lässt sich eine Funktion F auf "geeignete" Ausdrücke anwenden. Die bei der Funktionsanwendung einzuhaltenden syntaktischen Nebenbedingungen betreffen
(1) die Anzahl der Ausdrücke (Stelligkeit der Funktion),
(2) die Sorten der Ausdrücke (Funktionalität der Funktion).

Es ist zwischen der Funktionsanwendung $F(E_1, ..., E_n)$ und ihrem Wert $f(a_1, ..., a_n)$ zu unterscheiden. Der Wert der Funktionsanwendung, also f, wird auch einfach als „Funktion“ bezeichnet.

Die Syntax einer Funktionsanwendung lässt sich durch eine einfache BNF-Regel beschreiben, die in Interaktion 50 in ein äquivalentes Syntaxdiagramm umzuwandeln ist. Üblicherweise wird eine Funktionsanwendung in Präfixschreibweise mit geklammerter Folge von aktuellen Argumenten formuliert.

- $I_\beta[F(E_1, \ldots, E_n)] = \begin{cases} f(I_\beta[E_1], \ldots, I_\beta[E_n]) & \text{falls } \forall i,\ 1 \le i \le n : I_\beta[E_i] \neq \perp \\ \perp & \text{sonst} \end{cases}$
- f ist dabei die Interpretation des Funktionsausdrucks F unter der Belegung β, d.h. $f = I_\beta[F]$

- Auswertung eines Funktionsaufrufs
  - Call-by-Value (Wertaufruf)
    - zuerst werden in beliebiger Reihenfolge die Argumentausdrücke ausgewertet
  - Alternative: Call-by-Name (Namensaufruf)
    - es wird mit der Auswertung des Funktionsaufrufs begonnen, bevor die Parameterausdrücke ausgewertet sind

Information 51: Interpretation und Auswertung der Funktionsanwendung

Nach der Beschreibung der Syntax ist auch bei diesem Sprachelement wieder die Frage nach der Semantik zu stellen. Es ist also die Interpretationsabbildung I zu diesem Sprachelement festzulegen. Die Interpretation der Funktionsanwendung bedeutet die Anwendung der durch f beschriebenen Funktion auf die Werte der aktuellen Parameterausdrücke. f ist dabei die Interpretation des Funktionsausdrucks F unter der Belegung β, d.h. $f = I_\beta[F]$.

Die Auswertung des Funktionsaufrufs kann gemäß zwei grundsätzlich verschiedene Strategien erfolgen:
(1) Alle Argumentausdrücke müssen ausgewertet sein, bevor mit der Auswertung des Funktionsaufrufs begonnen wird (*Call-by-Value*, d.h. Funktionsaufruf mit dem Wert der Parameter).
(2) Die Argumentausdrücke müssen noch nicht ausgewertet sein (*Call-by-Name*).

Im Weiteren wird ausschließlich von der *Call-by-Value*-Auswertungsstrategie ausgegangen.

- Aufgabe eines Identifikators
  - hat die Funktion eines Platzhalters
    - steht für einen Wert, der später eingesetzt wird
  - bezeichnet ein bestimmtes, unter Umständen erst später genauer anzugebendes Element
    - Vergleich mit der Mathematik: „Sei x ein Element der Menge M mit dem Wert ...“

- Beispiel: x · (x+1)
  - Funktion, die eine natürliche Zahl auf eine andere abbildet
    f: $\mathbb{N} \rightarrow \mathbb{N}$
  - wird definiert durch die Gleichung
    f(x) = x·(x+1) für alle x $\in \mathbb{N}$
  - Notation in Programmiersprachen: Funktionsdeklaration

Information 52: Identifikatoren in Funktionen

Die Platzhalterfunktion von Identifikatoren drückt aus, dass diese Identifikatoren den Platz für einen später einzusetzenden Wert einnehmen. Identifikatoren weisen auch die Eigenschaft auf, dass sie im Allgemeinen konsistent gegen einen Identifikator mit anderer Bezeichnung ersetzt werden können. Das gilt genau dann, wenn durch die neue Bezeichnung keine Beziehungskonflikte auftreten. Im obigen Beispiel x·(x+1) führt die konsistente Ersetzung von x gegen y zu y·(y+1). Zwischen der Interpretation von x·(x+1) und y·(y+1) wird nicht weiter unterschieden.

Die nachfolgenden Ausführungen dienen als Vorbereitung zur Einführung des Sprachelements der Funktionsdefinition.

- Notation, bei der auf die Einführung einer Bezeichnung f für die Funktion verzichtet wird: (nat x) nat: x·(x+1)
  - x wird im Funktionsbereich gebunden genannt (im Gegensatz zu frei)
  - x ist syntaktisch durch die Kennzeichnung nat x als Platzhalter (formaler Parameter) ausgezeichnet

- Gebundene Identifikatoren können durch beliebige andere Identifikatoren konsistent ersetzt werden
  - Voraussetzung: keine Konflikte zu freiem Auftreten der betreffenden Bezeichnung im betrachteten Ausdruck
  - im Beispiel kann x gegen y ersetzt werden: (nat y) nat: y·(y+1)
  - Kann x gegen y in (nat x): x·(x+y) ersetzt werden?

- Diese Umbenennung gebundener Bezeichnungen heißt α-Konversion

Interaktion 53: Gebundene und freie Identifikatoren

Zunächst wird eine notationelle Änderung an der Funktionsschreibweise vorgenommen, indem auf die Funktionsbezeichnung f verzichtet wird, wie in Interaktion 53 am Beispiel (nat x) nat: x·(x+1) gezeigt wird.

Durch die Kennzeichnung nat x als Platzhalter wird x im Funktionsausdruck gebunden. Freie Identifikatoren sind in Funktionsausdrücken genau solche Identifikatoren, die keine formalen Parameter sind.

Gebundene Identifikatoren haben die Eigenschaft, dass sie konsistent ersetzt werden können. Allerdings muss man darauf achten, dass durch die Umbenennung keine Konflikte zu nicht gebundenen Auftreten der Bezeichnung entstehen, wie anhand des abgeänderten Beispiels in Interaktion 53 verdeutlicht wird.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Schreibweise für eine Funktionsabstraktion $(s_1\ x_1, \ldots, s_n\ x_n)\ s : E$
  - E ist ein Ausdruck der Sorte s
  - $x_1, \ldots, x_n$ heißen formale Parameter
  - $(s_1\ x_1, \ldots, s_n\ x_n)\ s$ heißt Kopfzeile der Funktionsabstraktion
    - die Kopfzeile gibt die Funktionalität der Funktionsabstraktion wieder
    - im Beispiel: $M_1 \times \ldots \times M_n \rightarrow M$
      - M_i bezeichnen dabei die Mengen der Elemente der Sorten s_i
  - der Ausdruck E heißt Rumpf der Funktionsabstraktion

- Eine Funktionsabstraktion wird in Verbindung mit einer Funktionsanwendung genutzt
  - $((s_1\ x_1, \ldots, s_n\ x_n)\ s : E)\ (E_1, \ldots, E_n)$

Information 54: Funktionsabstraktion

Die Bestandteile der Funktionsabstraktion sind:

- ein Ausdruck E, der den so genannten Rumpf (*Body*) der Funktionsabstraktion darstellt.
- eine Liste von formalen Parametern einschließlich deren Sorten und der Sorte des von E gelieferten Ergebnisses, die gemeinsam den Kopf (*Head*) oder die Kopfzeile ausmachen. Mit den Sorten sind entsprechende (Träger-) Mengen M_i verknüpft, die den zulässigen Wertebereich der Parameter und des Funktionsergebnisses festlegen.

Der Kopf gibt die Funktionalität der beschriebenen Funktion an. Da die Sorten Mengen sind, kann die Funktionalität auch in der Mengenschreibweise angegeben werden.

- Termersetzungsregel:
  $((s_1\ x_1, ..., s_n\ x_n)\ s: E)(E_1, ..., E_n) \rightarrow E[E_1/x_1, ..., E_n/x_n]$
  falls E_i in Normalform für $1 \leq i \leq n$
  - besagt: vollständig ausgewertete Ausdrücke auf der Argumentposition werden in den Rumpfausdruck der Funktionsabstraktion eingesetzt
  - Auswertung gemäß der Call-by-Value-Auffassung
  - erfüllt die Striktheitsregel

- Unbedingte Termersetzungsregel:
  $((s_1\ x_1, ..., s_n\ x_n)\ s: E)(E_1, ..., E_n) \rightarrow E[E_1/x_1, ..., E_n/x_n]$
  - Auswertung erfolgt durch Einsetzen der noch nicht auf Normalform gebrachten aktuellen Parameter
  - Auswertung gemäß der Call-by-Name-Auffassung
  - erfüllt nicht die Striktheitsregel

Information 55: Auswertung der Funktionsanwendung

Die Grundlage zur Auswertung der Funktionsanwendung liefern die Termersetzungsregeln. Die in Information 55 angegebene Termersetzungsregel besagt, dass die auftretenden x_i durch die E_i im Ausdruck E (der Rumpf der Funktion) zu ersetzen sind. Hierzu muss allerdings sichergestellt sein, dass in E_i nur primitive Funktionen auftreten. Diese Bedingung entspricht der Forderung, dass die E_i in Normalform sind. In der Bedingung und damit in der gesamten Termersetzungsregel ist enthalten, dass die Auswertung der Ausdrücke auf der Argumentposition vollständig ausgewertet sind, bevor sie in den Rumpf eingesetzt und dieser dann berechnet wird. Das entspricht genau der Auffassung *Call-by-Value*. Insbesondere wird durch diese Termersetzungsregel und somit durch die *Call-by-Value*-Regel die Striktheitsregel erfüllt.

Es ist aber auch vorstellbar, dass die Auswertung eines Ausdrucks E_i erst dann vorgenommen wird, wenn im Rumpf das x_i auftritt. Das führt zur *Call-by-Name*-Auswertung. Durch die Möglichkeit, dass *Call-by-Name* nicht-strikte Funktionen zulässt, entsteht auch der folgende gravierende Unterschied, dass die Auswertung nach der *Call-by-Name*-Regel häufiger terminiert als gemäß der *Call-by-Value*-Regel.

5 REKURSIVE FUNKTIONSDEKLARATION

Bislang eingeführte Sprachelemente lassen keine nichtterminierenden (also nicht endlich-beschränkte bzw. unendliche) Berechnungen zu. Nichtterminierung würde nur dann entstehen, wenn die Berechnung irgendeiner Funktion der Basissignatur nicht terminieren würde. Diese Annahme ist allerdings vor dem Hintergrund der tatsächlich angenommenen Basisoperationen, wie die arithmetischen Operationen oder auch die if-Funktion nicht aufrechtzuerhalten. Was lediglich passieren kann, ist die Rückgabe eines undefiniert-Wertes $\perp$, wie z.B. bei der Division durch 0.

Durch das Sprachelement der Rekursion wird diese Beschränkung der statisch beschränkten Länge der Berechnungssequenz aufgehoben, wodurch das Problem der Nichtterminierung auftreten kann [Mö03].

5.1 Rekursiver Java-Methodenaufruf

Das in Information 56 gezeigte Beispiel einer rekursive Funktion berechnet die Addition natürlicher Zahlen und führt (unter der Voraussetzung, dass die Forderung im Kopf der Funktion erfüllt wird) zu terminierenden Berechnungsfolgen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Bislang lassen sich nur Ausdrücke aufschreiben, die stets auf terminierende Berechnungen führen
  - Länge der Berechnungssequenzen damit stets statisch beschränkt
- Durch das Konzept der rekursiven Deklaration von Funktionen wird die Formulierung von Rechenvorschriften mit unbeschränkt langen Berechnungen möglich
- Beispiel: Addition zweier ganzer Zahlen
  - Basis: Rechenstruktur der natürlichen Zahlen mit den Operationssymbolen zero (abgek. 0), succ, pred und Prädikat ==

```
static int add (int x, int y /* x, y >= 0 */ ) {
  if (x==0) return y;
  else return succ(add(pred(x), y));
}
```

Information 56: REKURSIVE FUNKTIONSDEKLARATION – Einführung

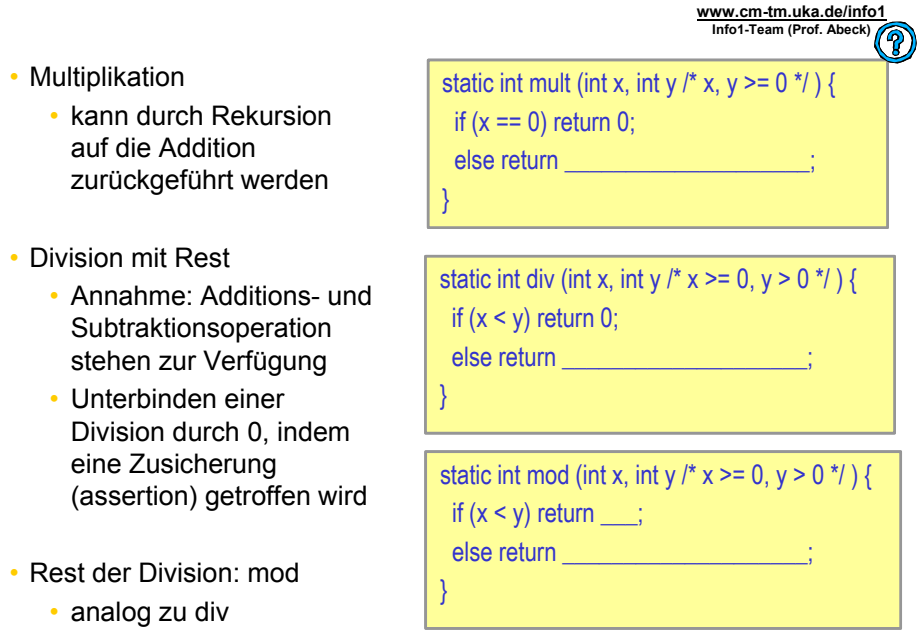
Die korrekte Arbeitsweise der rekursiven Funktionsdeklaration, durch die die Addition durchgeführt wird, kann man sich geeignet anhand eines Aufrufs mit konkreten Parameterwerten klarmachen.

Die rekursive Funktionsdeklaration ist in Java-Syntax angegeben. Da in Java die natürlichen Zahlen nicht als Zahlentyp zur Verfügung stehen, wird der ganzzahlige Typ int verwendet. Die Einschränkung auf die natürlichen Zahlen erfolgt im Beispielprogramm durch einen Kommentar in der Kopfzeile der Funktionsdeklaration.

Bereits an diesem einfachen Beispiel einer rekursiven Funktion lässt sich ein wichtiges Phänomen der Rekursion beobachten: Mit der Berechnung (Ausführung) der Nachfolgerfunktion succ kann erst dann begonnen werden, wenn man an das Ende der rekursiven Aufrufkette von add gelangt ist. Die Ausführung erfolgt dann in entgegen gesetzter Richtung zur Erzeugung der succ-Funktionen. Man nennt dieses Phänomen auch Nachklappern. Als Konsequenz muss die nachklappernde succ-Funktionen in bestimmter Form, und zwar gemäß dem Kellerprinzip, zwischengespeichert werden.

5.2 Weitere Beispiele

Ähnlich wie im vorhergehenden Beispiel die Nachfolger-Funktion zur rekursiven Berechnung der Addition verwendet wurde, kann die Multiplikation auf der Addition aufbauen. Die Arbeitsweise dieser in Interaktion 57 zu vervollständigenden, rekursiven Funktionsdeklaration kann man sich wiederum anhand eines Aufrufs mit konkreten Werten verdeutlichen.



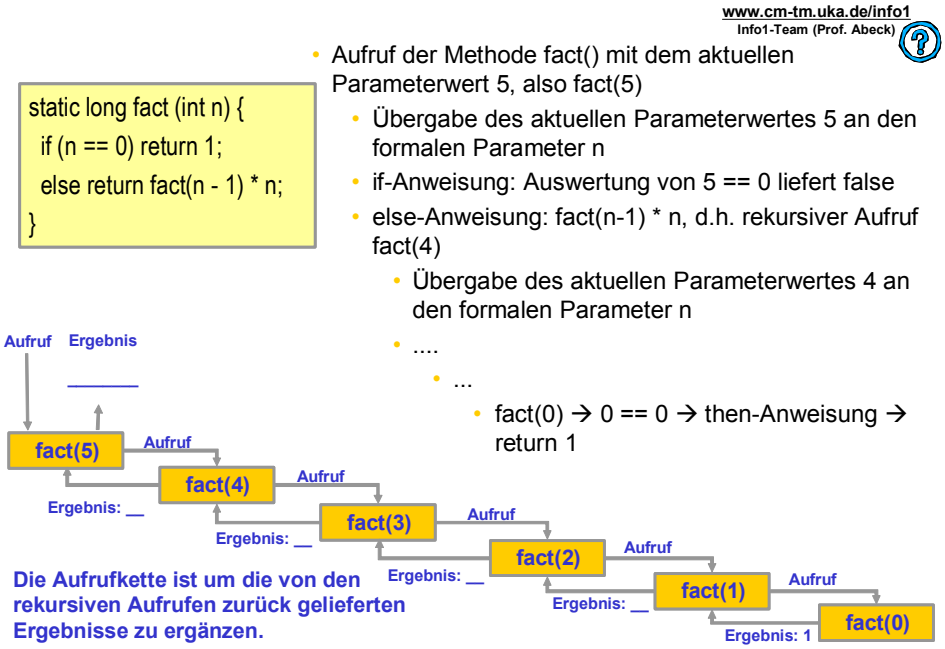
Interaktion 57: Multiplikation, Division mit Rest, Rest der Division

In diesem Fall ist nun die Addition die nachklappernde Funktion. Man kann sich leicht vorstellen, wie aufwendig diese Form der Realisierung der Multiplikation ist, da jede nachklappernde Addition eine Anzahl nachklappernder succ-Funktionen nach sich zieht.

Beim nächsten Beispiel, der Division mit Rest (auch ganzzahlige Division genannt, Infix-Schreibweise $x \div y$) muss eine Division durch 0 verhindert werden, weshalb im Kommentar $y > 0$ gefordert wird. Ein Verstoß gegen diese Bedingung führt zu einer nichtterminierenden Rekursion.

Das Gegenstück zur Division mit Rest (div) ist die Funktion, die den Rest der ganzzahligen Division berechnet, die modulo-Funktion (mod). Für diese Funktion mod gilt die gleiche Restriktion wie für div, dass der Nenner nicht 0 sein darf. Das Berechnungsprinzip ist einfach: Solange y noch in x hineingeht (x >= y), zieht man y von x ab und probiert es von neuem (rekursiver Funktionsaufruf). Im anderen Fall, also x < y ist der Wert x das Ergebnis.

Interaktion 58 greift das bereits kennen gelernte Beispiel der Fakultätsfunktion wieder auf und verdeutlicht den Programmablauf, der beim Aufruf von fact(5) entsteht.



Interaktion 58: Ablauf einer rekursiven Berechnung am Beispiel der Fakultät

Die Berechnung eines Aufrufs der Funktion fact() führt solange zu einem rekursiven Aufruf, bis der aktuelle Parameter, mit dem die Funktion aufgerufen wird, den Wert 1 erreicht hat. Da sich der Wert des aktuellen Parameters beim rekursiven Aufruf jeweils um 1 verringert, wird dieser Fall beim Aufruf von fact() mit einem Wert n > 0 nach n - 1 rekursiven Aufrufen erreicht.

Wie die in Interaktion 58 abgebildete Aufrufkette für den Aufruf von fact(5) skizziert, erfolgt nach Erreichen des Endes der Aufrufkette bei fact(1) die eigentliche Ergebnisberechnung, indem das zurück gelieferte Ergebnis des rekursiven Aufrufs in den Ausdruck fact(n - 1) * n eingesetzt wird. D.h., fact(1) liefert 1 zurück, das von fact(2) zur Berechnung des Ergebnisses von 1 * 2 = 2 nutzt, womit fact(3) seinerseits das Ergebnis 2 * 3 = 6 zurückliefern kann, usw.

Im Falle der Methode fact() handelt es sich um eine so genannte direkte Rekursion, weil sich die Methode direkt selbst aufruft. Liegen Methodenaufrufe zwischen dem rekursiven Aufruf, spricht man von indirekter Rekursion.

- Jedes rekursiv gelöste Problem kann auch iterativ gelöst werden und umgekehrt
  - iterative Lösungen sind im Prinzip kostengünstiger, weil Funktions- bzzw. Methodenaufrufe laufzeit- und speicherintensiv sind
  - rekursive Lösungen können einfach er zu verstehen und kürzer in der Aufschreibung sein

- Rekursionen treten an vielen Stellen in der Informatik auf, wie z.B.
  - bei der der Beschreibung von formalen Sprachen
  - Deklaration von nichtterminalen Hilfsbezeichnungen für BNF-Ausdrücke
  - rekursive Datenstrukturen

Information 59: Abschließende Bemerkungen zur Rekursion

Das behandelte Fakultätsberechnungsproblem könnte auch iterativ, d.h. in Form einer Schleife gelöst werden. Dieser Sachverhalt gilt grundsätzlich für beliebige rekursive Lösungen, was eine wichtige und beweisbare Aussage der Berechenbarkeitstheorie darstellt.

Die Rekursion gehört zu den Kernelementen der Informatik. Wie die in Information 59 angegebenen Beispiele zeigen, tritt die Rekursion nicht nur in Programmiersprachen bei der rekursiven Deklaration von Rechenvorschriften auf, sondern auch im Zusammenhang mit formalen Sprachen und (rekursiven) Datenstrukturdefinitionen, wie in der Kurseinheit OBJEKTORIENTIERTE PROGRAMMIERUNG verdeutlicht wird.

VERZEICHNISSE

Abkürzungen und Glossar

Abkürzung oder Begriff	**Langbezeichnung und/oder Begriffserklärung**
BNF	Backus Naur Form Schreibweise für Regeln einer Grammatik.
Call-by-Value	Strategie beim Aufruf einer Funktion, die vorschreibt, dass alle Argumentausdrücke ausgewertet sein müssen, bevor mit der Auswertung des Funktionsaufrufs begonnen wird. Deutscher Begriff: Wertaufruf
Call-by-Name	Strategie beim Aufruf einer Funktion, bei der nicht alle Argumentausdrücke ausgewertet sein müssen, bevor mit der Auswertung der Funktion begonnen wird. Deutscher Begriff: Namensaufruf
dyadisch	2-stellig Beispiel: Addition (+) ist ein dyadischer Operator
ENV	*Environment* Bezeichnet im Zusammenhang mit der Semantik-Festlegung von Identifikatoren die Menge aller Belegungen von Identifikatoren.
Funktionsanwendung	Das beherrschende Sprachelement der funktionalen Programmiersprachen. Synonymer Begriff: Funktionsapplikation
Grundterm	Ein Term, der ausgehend von den 0-stelligen Funktionen einer Signatur induktiv über den Termaufbau gebildet werden kann.
Modus Ponens	Bezeichnung einer Ableitungsregel der Aussagenlogik, die besagt, dass bei Kenntnis der Prämisse eines zulässigen Schlusses auch das Resultat dieses Schlusses abgeleitet werden kann.
monadisch	1-stellig Beispiel: Negation (-) ist ein monadischer Operator
Nachklappern	Bezeichnung eines Phänomens, das im Zusammenhang mit der verzögerten Berechnung von rekursiven Aufrufen entsteht.
Normalform (Term)	Ein Term ist in Normalform, wenn er ein terminaler Grundterme zu einem Termersetzungssystem R ist.
Rechenstruktur	Zusammenfassung von Trägermengen und Operationen. Äquivalenter mathematischer Begriff: Algebra
Rekursion	Zurückführen der Lösung eines Problems auf das gleiche Problem.

Signatur	Legt fest, in welcher Weise die Funktionssymbole einer Rechenstruktur mit den Sorten sinnvoll verknüpft werden können.
Signaturdiagramm	Graphische Darstellung der Signatur einer Rechenvorschrift.
stark	Eigenschaft einer Funktion, die für beliebige Argumente stets die Resultate L oder O liefert, also niemals ⊥ als Resultat hat. Englischer Begriff: *strong*
strikt	Eigenschaft einer Rechenstruktur, die sicherstellt, dass aus der Undefiniertheit von nur einem Argument die Undefiniertheit des Ergebnisses der Rechenstruktur folgt.
Substitution	Formaler Mechanismus zum Arbeiten mit den Identifikatoren (insbesondere zur Wertebelegung).
Term	Ein gemäß einer vorgegebenen Signatur korrekt gebildeter Ausdruck.
Termersetzungssystem	Ein über einer Signatur Σ im Allgemeinen endliche Menge R von Termersetzungsregeln.
Tertium non datur	Bezeichnung einer Ableitungsregel der Aussagenlogik, die besagt, dass es einen dritten Wert (neben O und L) nicht gibt.
wff	well formed formula Formel, die vorgegebenen Termaufbaugesetzen genügt.
⊥	*Bottom*-Element Symbolisiert im Zusammenhang mit Zuständen den gedachten „Endzustand“ nichtterminierender Programme. Synoymer Begriff: Undefiniert-Element

Index

Informationen und Interaktionen

Literatur

[Br98] Manfred Broy: Informatik – Eine grundlegende Einführung, Band 1: Programmierung und Rechstrukturen, Springer Verlag 1998.

[Go97] Gerhard Goos: Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer Verlag 1997.

[Mö03] Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

Kursblock

PROGRAMMIERKONZEPTE

In diesem Kursblock werden das algorithmische Denken und die Umsetzung von Algorithmen in lauffähige Java-Programme vermittelt. Die ersten zwei Kurseinheiten behandeln die imperativen und objektorientierten Programmierkonzepte. Abschließend werden mit der Ausnahmebehandlung und den *Threads* zwei durch die Sprache Java unterstützte, fortgeschrittene Programmierkonzepte vorgestellt.

Der Kursblock setzt sich aus den folgenden drei Kurseinheiten zusammen:

IMPERATIVE PROGRAMMIERUNG

Kurzbeschreibung

Ausgehend von den Variablen und Zuweisungen werden in dieser Kurseinheit Sprachelemente und Datenstrukturen der imperativen Programmierung eingeführt. Aspekte wie Zuweisungssemantik, Zusicherung, Schleifeninvarianten und Effizienz werden behandelt.

Schlüsselwörter

Variable, Verzweigung, Schleife, Zusicherung, Effizienz, Wertemenge, Array, mehrdimensionales Array, Zeichen, Zeichenkette, String, Stringoperation

Lernziele

1. Das Konzept der Variablen und Zuweisungen als Kern der imperativen Programmierung wird verstanden.
2. Die wichtigsten Anweisungen imperativer Programmierung sowie der Methodenaufruf können zur Erstellung eigener Programme genutzt werden.
3. Zusicherungen und Schleifeninvarianten können zu einem imperativen Programm formuliert werden.
4. Datenobjekte vom Typ Array bzw. String können innerhalb der imperativen Programmierung deklariert und verwendet werden.

Hauptquellen

- Manfred Broy: Informatik – Eine grundlegende Einführung, Band 1: Programmierung und Rechenstrukturen, Springer Verlag 1998.
- Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

Inhaltsverzeichnis

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- VARIABLEN UND ZUWEISUNGEN
  - Wiederverwendung von Identifikatoren, Zustandsmenge, Semantik
- ZUSAMMENGESETZTE ANWEISUNGEN
  - Verzweigungen, Zusicherungen, Schleifen, Schleifeninvarianten
- METHODEN
  - Überladen, Problemzerlegung mittels Methoden
- DATENSTRUKTUREN
  - Arrays, Strings

Information 1: IMPERATIVE PROGRAMMIERUNG

1 VARIABLEN UND ZUWEISUNGEN

In dieser Kurseinheit werden die konzeptionellen Grundlagen der imperativen Programmierung eingeführt und am Beispiel der Programmiersprache Java praktisch umgesetzt. Die Ausführungen zu den Variablen und Zuweisungen orientieren sich an [Br98], während die Java-orientierten Inhalte an [Mö03] angelehnt sind.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Imperative oder zuweisungsorientierte Programme: Folge von Anweisungen
- Anweisungen dienen u.a. zur Kontrolle und Steuerung von Abläufen
- Durch die Ausführung einer Anweisung ändert sich der Zustand
- Der Zustand betrifft zwei Aspekte
  - Datenzustand und Ablaufzustand
- Anweisungen werden häufig in Abhängigkeit vom Speicher- und Ablaufzustand erteilt
- Anweisungen ändern gewisse Teile des Speicher- und Ablaufzustands

Information 2: VARIABLEN UND ZUWEISUNG-
Zentraler Begriff der Anweisung

Der imperativen Programmierung liegt ein operativer und maschinenorientierter Zugang zur Programmierung zugrunde. Dieses Programmierkonzept hat sich in der Praxis gegenüber dem in der Kurseinheit RECHENSTRUKTUREN UND FUNKTIONALE PROGRAMME behandelten Konzept der funktionalen Programmierung durchgesetzt. Auf der imperativen Programmierung setzt die in einer weiteren Kurseinheit behandelte OBJEKTORIENTIERTE PROGRAMMIERUNG auf.

Die Vorgänge, die Maschinenzustandsübergänge bewirken, werden als Anweisungen bezeichnet. Information 2 beinhaltet zentrale Aussagen über Anweisungen.

Es wird deutlich, dass der Zustand bei der imperativen Programmierung eine ganz zentrale Rolle einnimmt. In der Kurseinheit GRUNDBEGRIFFE DER INFORMATIK wird der Zustand durch den Speicherinhalt des Rechensystems definiert. Bei näherer Analyse, was den Zustand ausmacht, wird eine Zweiteilung sichtbar: der Speicher- oder Datenzustand und der Ablauf- oder Kontrollzustand. Der Teil des Speichers, der den Ablaufzustand beinhaltet, wird als Laufzeitkeller bezeichnet.

In bestimmten Fällen soll eine Anweisung nur dann erteilt werden, wenn der zu ändernde Ausgangszustand bestimmte Eigenschaften aufweist. Hierzu besteht die bedingte Zuweisung, die in ihrem Bedingungsteil solche Eigenschaften abzuprüfen erlaubt.


```
Statement =     {Ident ":"}
                (  Block
                |  [Expr] ";"
                |  "if" "(" Expr ")" Statement ["else" Statement]
                |  "switch" "(" Expr ")" "{" {switchGroup} "}"
                |  "while" "(" Expr ")" Statement
                | "for" "(" [ForInit] ";" [Expr] ";" [ForUpdate] ")" Statement
                | "do" Statement "while" "(" Expr ")"
                | "try" Block (Catch {Catch} [Finally] | Finally )
                | "synchronized" "(" Expr ")" Block
                | "return" [Expr] ";"
                |  "throw" Expr ";"
                |  "break" [Ident] ";"
                |  "continue" [Ident] ";"
                ).
```

Information 3: Anweisungen der Sprache Java im Überblick

Information 3 führt die Anweisungen, die in der Sprache Java bereitgestellt werden, in Erweiterter Backus-Naur-Form (EBNF) auf.

Einige grundlegende Anweisungen, wie z.B. die if- oder while-Anweisung, sind bereits aus der Kurseinheit PROGRAMMIERGRUNDLAGEN bekannt. Verschiedene der aufgeführten Anweisungen werden in den beiden anderen Kurseinheiten dieses Kursblocks (OBJEKTORIENTIERTE PROGRAMMIERUNG und FORTGESCHRITTENE PROGRAMMIERKONZEPTE) behandelt.

1.1 Motivation von Variablen

Ein offensichtliches Ziel, das mit gebundenen Bezeichnern erreicht werden kann, besteht darin, Zwischenergebnisse mehrfach verwenden zu können. Hierin können Zwischenergebnisse und somit bereits geleistete Berechnungsarbeit gespeichert werden. Das führt zu einer effizienteren Arbeitsweise, weil dadurch die Arbeit gespart wird, das Zwischenergebnis von neuem berechnen zu müssen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Mit den gebundenen Bezeichnungen verknüpftes Ziel
  - Mehrfachverwendung von Zwischenergebnissen
- Berechnung des Ausdrucks E1 * E1 - E2 * E2
  - klassische Notation der Funktionsabstraktion
    ((int a, b) int: a * a - b * b) (E1, E2)
  - Programm mit Deklarationen (Konstanten, statische Variablen)
    - Konstanten werden in Java durch das Schlüsselwort "final" gekennzeichnet

```
final int a = 17 - 4 * 3;          // E1
final int b = (9 - 5) * 2;         // E2
result = a * a - b * b;
```

Die Deklaration der Größe result
ist im Java-Programm zu ergänzen

Interaktion 4: Gebundene Bezeichnungen

Dieser Sachverhalt wird in Interaktion 4 an einem einfachen Beispiel aufgezeigt. Das Ziel bei der Berechnung des Ausdrucks (*Expression*) E1 * E1 - E2 * E2 besteht darin, dass jeder der Ausdrücke E1 und E2 nur einmal ausgewertet werden soll. Erreicht wird dieses Ziel durch die Funktionsabstraktion ((int a, int b) int: a * a – b * b) (E1, E2). Das gleiche Ergebnis wird durch Elementdeklarationen erzielt, hier Deklaration der Konstanten a und b.

Das Java-Programm zeigt das Vorgehen am Beispiel von zwei einfachen Ausdrücken. Der in diesem Programm verwendete Bezeichner result ist in Interaktion 4 geeignet zu deklarieren.

Das Problem der Mehrfachberechnung von Teilausdrücken tritt beispielsweise ganz massiv bei der Berechnung von Polynomen auf. Bei traditioneller Berechnung der einzelnen Ausdrücke $a_i x^i$ führt man zwangsläufig eine unnötige Mehrfachberechnung durch, d.h. man muss mehrfach jeweils x^{i-1} ausrechnen. Hier schafft das in Information 5 beschriebene so genannte Hornerschema Abhilfe.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Bei folgendem Ausdruck (Polynom) müssen die x^i mehrfach berechnet werden:

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

- Umgehung der Mehrfachberechnung durch das Hornerschema

$$(...(a_n \cdot x + a_{n-1}) \cdot x + ... + a_1) \cdot x + a_0$$

- Aufbrechen der Formel, die durch das Hornerschema vorgegeben ist

$y_{n+1} = 0$
$y_n = y_{n+1} \cdot x + a_n$
$y_{n-1} = y_n \cdot x + a_{n-1}$
...
$y_0 = y_1 \cdot x + a_0$

- Die Wiederverwendung von Hilfsidentifikatoren führt zu den Variablen

Information 5: Hornerschema als weiteres Beispiel

Eine direkte Übertragung des Hornerschemas mit einer entsprechenden Folge von Vereinbarungen führt zu einer Berechnungsfolge, wie sie in Information 5 angegeben ist. Das Ergebnis ist ein gestaffeltes System von Deklarationen.

Es fällt auf, dass jeder Hilfsidentifikator y_i nach seiner Einführung ausschließlich in der darauf folgenden Zeile verwendet wird. Es liegt also nahe, einen einmal eingeführten Identifikator wieder zu verwenden.

Zur Umsetzung des Hornerschemas wird nur ein Identifikator benötigt. Dieser Identifikator wird mit wechselnden, also variablen Werten belegt, weshalb er als Variable oder Programmvariable bezeichnet wird. Eine Anweisung der Form v=E heißt Zuweisung.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
final int x = 3; final int a4 = 2; final int a3 = 1; final int a2 = 3;  final int a1 = 4; final int a0 = 6;
int v = 0;              // declaration of variable v
v = v*x + a4;
v = v*x + a3;
v = v*x + a2;
v = v*x + a1;
v = v*x + a0;
// Polynomial:          _____________________________
// Result:              _________
```

- v heißt Programmvariable, v = E; ist eine Zuweisung (spezielle Form einer Anweisung)

- Syntax (Ausschnitt)

```
Assignment      =       AssignmentOp Expr.
AssignmentOp    =       "=" | "+=" | "-=" | "/=".
```

Interaktion 6: Variablen und Zuweisungen

In Interaktion 6 wird das Java-Programm angegeben, durch das die Berechnung eines Polynoms gemäß dem vorgestellten Hornerschema erfolgt. Dabei wird ein festes n = 4 angenommen. Nach der Deklaration der Variablen v folgen entsprechende Zuweisungen (*Assignments*), durch die das Hornerschema realisiert wird. Das durch das Programm berechnete Polynom und das Resultat sind als Kommentar im Programm zu ergänzen.

Wie die in Interaktion 6 angegebene Syntaxbeschreibung von Zuweisungen in Java zeigt, sind neben dem Zuweisungsoperator = weitere Zuweisungsoperatoren definiert, durch die sich die Zuweisung mit einer arithmetischen Operation kombinieren lässt.

1.2 Semantik von Zuweisungen

Im obigen Beispiel kommt die Variable v in der Berechnung sowohl auf der linken als auch auf der rechten Seite vor. Betrachtet man das Konzept der Variable und Zuweisung aus der Sicht der klassischen Mathematik, so stößt man bei der Interpretation auf Schwierigkeiten.

Offensichtlich ist bei einer durchaus zulässigen Zuweisung

$$x = x+1$$

die Interpretation des Zuweisungszeichens als Gleichheitszeichen mathematisch nicht möglich. Vielmehr muss die Interpretation in folgender Weise unter Zuhilfenahme des Gleichheitszeichens erfolgen:

$$x_{neu} = x_{alt} + 1$$

Die Größe x existiert somit in zwei Zuständen mit zwei Werten: der Wert von x im alten Zustand (x_{alt}) und der Wert von x im neuen Zustand (x_{neu}).

An dieser Stelle wird erneut der Sachverhalt deutlich, dass Anweisungen – in diesem Fall Zuweisungen – Zustände ändern.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Im Zusammenhang mit Anweisungen betrachtete Zustände
  - Belegungen der Identifikatoren, die als Programmvariablen auftreten
    - wird als Menge ENV bezeichnet
  - Hinzufügen eines speziellen Zustands $\perp$
    - führt insgesamt zu einer Menge STATE = ENV $\cup$ {$\perp$}
  - Definition einer partiellen Ordnung $\sqsubseteq$ auf STATE
- Funktionale Bedeutung der Ausführung einer Anweisung
  - Änderung eines Zustands
  - Interpretationsfunktion I: <statement> $\rightarrow$ (STATE $\rightarrow$ STATE)
    - Zustandsabbildung: Zustand vor Anweisung $\rightarrow$ Zustand nach Anweisung
- Operationale Semantik von Anweisungen
  - wird üblicherweise nicht durch (sehr komplexe) Termersetzungssysteme, sondern durch abstrakte Maschinenmodelle beschrieben

Information 7: Funktionale Bedeutung von Anweisungen

Um die Semantik von Anweisungen festlegen zu können, muss zunächst geklärt sein, was formal unter einem Zustand zu verstehen ist. Das führt zur Werte-Belegung der die Variablen

darstellenden Identifikatoren und zu der Menge ENV (steht für *Environment*). Dieser Menge wird das so genannte *Bottom*-Symbol oder Undefiniert-Symbol $\perp$ hinzugefügt.

Die Zustandsdefinition wird dazu genutzt, die funktionale Bedeutung von Anweisungen anzugeben: Als Semantik (Interpretationsfunktion) einer Anweisung wird eine Zustandsabbildung betrachtet, die für jeden Anfangszustand (Zustand vor der Ausführung der Anweisung) einen Endzustand (Zustand nach Ausführung der Anweisung) erzeugt.

Dieses Vorgehen basierend auf den Zustandsabbildungen ließe sich weiterführen, um die operationelle Semantik von Anweisungen zu beschreiben. Hierzu wäre notwendig, die „Rechenstruktur der Zustände“ in allen Einzelheiten zu beschreiben und dafür ein Termersetzungssystem anzugeben. Aufgrund der Komplexität geht man in der Regel einen anderen Weg, der über die abstrakten Maschinenmodelle führt. Hier treten die Kontroll- und Speicherzustände im Gegensatz zu den Termersetzungssystemen explizit auf, was den Konstruktionsprozess erheblich vereinfacht.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

$$I[x = E](\sigma) = \begin{cases} \sigma[I_\sigma[E]/x] & \text{falls } \sigma \neq \perp \wedge I_\sigma[E] \neq \perp \\ \perp & \text{falls } \sigma = \perp \vee I_\sigma[E] = \perp \end{cases}$$

- Zuweisung x = E bewirkt eine Zustandsänderung von einem Ausgangszustand in einen Nachfolgezustand
  - Ausgangszustand stimmt mit dem Nachfolgezustand für alle Identifikatoren bis auf x überein
  - Bedeutung von $\sigma[I_\sigma[E]/x]$: Im Nachfolgeszustand liefert x den Wert von E
- Nachfolgezustand ist undefiniert, wenn
  - Ausgangszustand undefiniert ist
  - Semantik von E undefiniert ist

Information 8: Semantik einer Zuweisung

An dieser Stelle wird die Semantik der in der imperativen (zuweisungsorientierten) Programmierung zentralen Anweisung, der Zuweisung, näher betrachtet.

Allgemein lässt sich die Semantik der Zuweisung durch eine in Information 8 angegebene Interpretationsvorschrift formulieren. Diese besagt, dass bei einem Zustand σ die Ausführung der Anweisung x = E zu einem Folgezustand führt, bei dem jedes x gegen $I_\sigma[E]$ ersetzt wird.
Das gilt allerdings nur für den Fall, dass weder σ noch $I_\sigma[E]$ dem undefinierten Zustand entsprechen. In diesem Fall liefert die Interpretationsvorschrift als Folgezustand der Zuweisung ebenfalls den undefinierten Zustand $\perp$. Allgemein bestehen imperative Programme aus einer Folge von Zustandsänderungen, die einer Folge von Zuweisungen entsprechen.

2 ZUSAMMENGESETZTE ANWEISUNGEN

Neben den einfachen Anweisungen (wie z.B. die im letzten Kapitel behandelte Zuweisung) bestehen die zusammengesetzten Anweisungen, die aus den einfachen Anweisungen durch verschiedene Formen der Komposition hervorgehen [Br96, Mö03].

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Durch verschiedene Formen der Komposition (Sequenz, Bedingung, Wiederholung) lassen sich aus gegebenen Anweisungen zusammengesetzte Anweisungen erzeugen
  - werden auch als sequentielle Komposition bezeichnet
- Syntax

  SequentialComposition = Statement ";" {Statement ";"}.
- Semantik

  $I[S1; S2](\sigma) =_{def} I[S2](I[S1](\sigma))$
- Anweisungen werden nacheinander (sequentiell) ausgeführt

Information 9: ZUSAMMENGESETZTE ANWEISUNGEN - Syntax und Semantik

Eine Form der Komposition ist die Sequenz oder Aufeinanderfolge. Die Syntax sieht vor, die aufeinander folgenden Anweisungen durch einen Strichpunkt (;) voneinander zu trennen. Semantisch bedeutet die sequentielle Komposition, dass der durch die zuerst aufgeschriebene Anweisung S1 erzeugte Folgezustand der Zustand ist, auf dem die zweite Anweisung S2 aufsetzt.

2.1 Verzweigungen

Eine wichtige zusammengesetzte Anweisung ist die bereits in den PROGRAMMIERGRUNDLAGEN eingeführte bedingte Anweisung, durch die eine Zweiweg-Verzweigung realisiert wird.

- Aufteilung des Programmflusses in zwei oder mehr Zweige
  - Zweiweg-Verzweigung: if-Anweisung
  - Mehrweg-Verzweigung: switch-Anweisung

- Semantische Interpretation der if-Anweisung

$$I[\text{if (C) S1; else S2;}](\sigma) =_{def} \begin{cases} I[S1](\sigma) & \text{falls } \sigma \neq \perp \text{ und } I_\sigma[C] = L \\ I[S2](\sigma) & \text{falls } \sigma \neq \perp \text{ und } I_\sigma[C] = 0 \\ \perp & \text{sonst} \end{cases}$$

- Kurzschlussauswertung

```
if (y != 0 && x / y > 10) ...;
/* if first comparison y != 0 is false (i.e. y is 0)  the second comparison
x / y > 10 is not evaluated */
```

Information 10: Verzweigungen

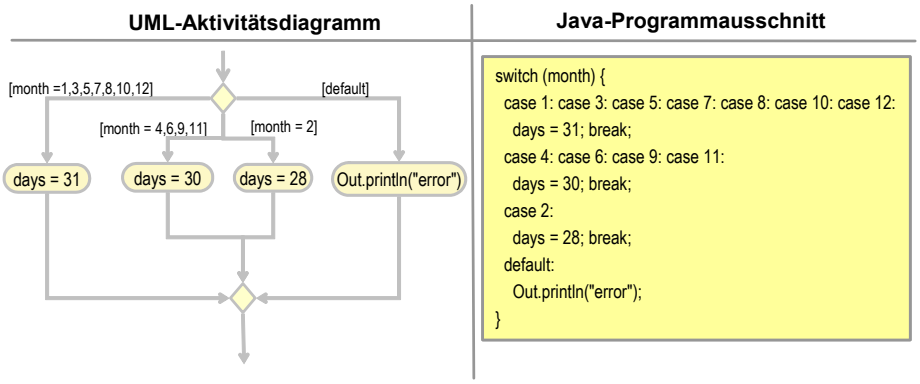
Bedingte Anweisungen werden in ihrer Bedeutung auf die Fallunterscheidung zurückgeführt. Hat der Ausdruck C in dem vor der Ausführung der bedingten Anweisung bestehenden Zustand den Wert true, so wird S1 ausgeführt, andernfalls S2.
Ist der Wert von C für den Anfangszustand undefiniert ($\perp$), also weder true noch false, so ist der Endzustand der bedingten Anweisung der undefinierte Zustand $\perp$.

Der Programmausschnitt in Information 10 zeigt, wie diese zu einem Laufzeitfehler führende Situation vermieden werden kann. Das Beispiel nutzt dabei die von Java durchgeführte Kurzschlussauswertung – auch als bedingte Auswertung bezeichnet – aus: In der im Beispiel auftretenden Und-Verknüpfung y != 0 && x / y > 10 wird der zweite Vergleich x / y > 10 gar nicht mehr ausgewertet, wenn der erste Vergleich y != 0 den Wert false ergibt.

Besteht aufgrund des umzusetzenden Algorithmus die Anforderung, den Programmfluss an einer Stelle in mehrere Zweige zu spalten, bietet sich die Mehrweg-Verzweigung in Form der switch-Anweisung an.

- switch-Anweisung prüft den Wert eines Ausdrucks
  - Ausdruck kann vom Typ int, short, byte oder char sein
  - schlägt in Abhängigkeit davon einen von mehreren möglichen Wegen ein

- Beispiel:



Information 11: Mehrwegverzweigung switch-Anweisung

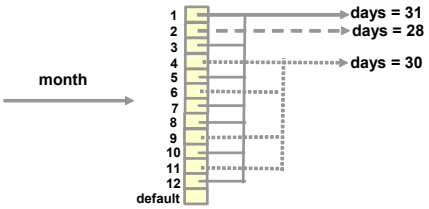
In Information 11 wird die Syntax der switch-Anweisung durch ein einfaches Beispiel, das die Anzahl der Tage (Variable days) eines Monats (Variable month, wobei die Monate Januar bis Dezember von 1 bis 12 durchnummeriert sind).

Ist in month z.B. der Wert 1 gespeichert, so steht dieser Wert für den Monat Januar. Die switch-Anweisung wählt den ersten Zweig aus (wegen case 1) und weist der Variablen days den Wert 31 zu (weil der Januar 31 Tage hat). Durch die break-Anweisung wird danach an das Ende der switch-Anweisung gesprungen.
Würde die break-Anweisung fehlen – ein häufiger Programmierfehler im Zusammenhang mit der switch-Anweisung – käme es zur Bearbeitung der folgenden case-Anweisungen und damit zu einem fehlerhaften Programmverhalten. Die break-Anweisung ist eine offensichtliche, von der Sprache C geerbte Schwachstelle der Sprache Java und wurde in anderen Sprachen (z.B. C#) behoben.

```
if (month == 1 || month == 3 || ...) days = 31;
else if (month == 4 || month == 6 || ...) days = 30;
else if (month == 2) days = 28;
else Out.println("error");
```

- Jede switch-Anweisung kann in eine semantisch gleichwertige Folge von if-Anweisungen übertragen werden

- Der vom Compiler erzeugte Code ist unterschiedlich
  - if-Anweisung: sequentielle Prüfung der Fälle
  - switch-Anweisung: jeder Fall ist ein Eintrag in einer vom Compiler angelegten Tabelle
    - Woraus besteht ein Tabelleneintrag?



Interaktion 12: switch-Anweisung und if-Anweisung

Offensichtlich lässt sich eine switch-Anweisung problemlos in eine if-Anweisung überführen, wie an dem Beispiel in Interaktion 12 aufgezeigt wird. Die Programmausschnitte der if-Anweisung und der in Information 11 angegebenen switch-Anweisung sind zwar semantisch gleichbedeutend, werden aber vom Compiler ganz unterschiedlich behandelt: Eine switch-Anweisung wird nicht wie eine if-Anweisung sequentiell durchlaufen, sondern mittels einer Tabelle abgearbeitet, deren Einträge auf den dazu gehörigen Zweig verweist.

- Die Lösung mittels switch-Anweisung
  - hat im Mittel eine kürzere Ausführungszeit als die Lösung mittels if-Anweisung
  - verbraucht im Mittel mehr Speicherplatz als die Lösung mittels if-Anweisung

- Ausführungszeit und Speicherplatz sind die zwei konkurrierenden Optimierungskriterien von Algorithmen

- Bei weit auseinander liegenden case-Marken entstehen große Tabellen
  - if-Anweisung in diesem Fall angemessener
  - gute Compiler übersetzen in einem solchen Fall die switch-Anweisung wie eine if-Anweisung

Information 13: Eigenschaften der beiden Lösungen

Durch die Einführung der Tabelle wird bei der Umsetzung der switch-Anweisung im Mittel der zu bearbeitende Zweig schneller als bei der Lösung mittels if-Anweisung erreicht. Dieser Vorteil der besseren Laufzeit wird erkauft durch einen im Mittel größeren Speicherplatzverbrauch, der durch die Tabelle verursacht wird.

Durch Zusicherungen (*Assertion*) lassen sich explizite Aussagen über den Programmzustand treffen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Zusicherungen (engl. Assertions) machen eine Aussage über den Zustand (STATE) des Programms an einer Programmstelle
  - können in Programmen als Kommentare geschrieben werden
  - in diesem Fall keine Auswertung durch den Compiler

- Beispiel: Zusicherungen zum then- und else-Fall

```
int x;
...
x = ...;
...
if (0 <= x && x < 10)
  // assertion:        0 <= x && x < 10
  //                   value of x is in [0 .. 9]
else
// assertion:         !(0 <= x && x < 10)
//                    ! (0 <= x) || !(x<10)  (DeMorgan)
//                    (x < 0) || (x >= 10)
```

Information 14: Zusicherungen

Im ersten Kapitel dieser Kurseinheit wurde der Zustand STATE eingeführt. Hierüber wurde die Interpretationsfunktion I und damit die Semantik der Anweisung definiert.

Wie das Beispiel in Information 14 zeigt, können so genannte Zusicherungen als Kommentare in das Programm eingefügt werden. Eine Auswertung durch den Compiler erfolgt dabei allerdings nicht.

Im Zusammenhang mit der if-Anweisung können bei der Zusicherung zum else-Fall die Regeln von DeMorgan hilfreich sein, um das Negationszeichen zu eliminieren.

2.2 Wiederholungsanweisungen (Schleifen)

In vielen Programmier-Situationen wünscht man die Ausführung einer Anweisung S, bis der erreichte Zustand eine gewisse Bedingung erfüllt. Die Anweisung, durch die diese Art von Aufgabe gelöst wird, ist die Wiederholungsanweisung. Neben der in den PROGRAMMIERGUNDLAGEN bereits behandelten while-Anweisung (Abweisschleife) stehen in Java mit der do-while-Anweisung (Durchlaufschleife) und der for-Anweisung (Zählschleife) zwei weitere Schleifenformen zur Verfügung.

- Durch die while-Anweisung wird eine Abweisschleife realisiert

- do-while-Anweisung (Durchlaufschleife)
  - im Unterschied zur while-Schleife wird der Rumpf mindestens einmal durchlaufen

- Beispiel:

```
int n = In.readInt();
if (n < 0) n = -n;        // no negative values
do {
  Out.print(n % 10);      // rest from n / 10
  n = n / 10;
}
while (n > 0);
```

Eingabe: n = 123 **Ausgabe: _____**

n % 10	n	Durchlauf

Was leistet das Programm? _____

Interaktion 15: do-while-Anweisung

Anhand des in Interaktion 15 angegebenen Beispiels kann man sich leicht klar machen, dass es Aufgabenstellungen gibt, in denen in jedem Fall ein Durchlauf durch die Schleife stattfinden sollte. Falls im obigen Beispiel eine while-Anweisung gewählt worden wäre, würde der Algorithmus für die Eingabe 0 nicht korrekt arbeiten.

Die dritte Schleifenvariante, die Zählschleife, wird häufig dann verwendet, wenn die Anzahl der Schleifendurchläufe im Voraus bekannt ist.

- Die eine Zählschleife realisierende for-Anweisung besteht aus den folgenden Teilen:
  (1) Initialisierungsteil
    - wird vor dem Betreten der Schleife ausgeführt
    - Laufvariable wird initialisiert
  (2) Abbruchbedingung
    - wird jedes Mal vor Betreten der Schleife überprüft
  (3) Inkrementierungsteil
    - wird am Ende jedes Schleifendurchlaufs ausgeführt
    - typische Verwendung: Erhöhen der Laufvariable
  (4) Schleifenrumpf

- Syntax
  "for" "(" Initialisierungsteil ";" Abbruchbedingung ";" Inkrementierungsteil ")"
  Schleifenrumpf ";"

Information 16: for-Anweisung

Die Auflistung der Bestandteile einer for-Anweisung in Information 16 deutet bereits an, dass es sich hierbei um eine komplexere Anweisung handelt, die Java zur Verfügung stellt.

Es ist davon abzuraten, die von Java zugelassenen, umfangreichen Möglichkeiten der for-Schleife auch tatsächlich zu nutzen, da hierdurch die Lesbarkeit erschwert wird. Sinnvoll ist die Beschränkung des Initialisierungs- und Inkrementierungsteils auf eine Anweisung. Komplexere Schleifen sollten deshalb durch eine while-Anweisung realisiert werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Programmbeschreibung:
  - Eingabe: positive ganze Zahl n
  - Ausgabe: (n x n) - Matrix m mit m[i, j] = i * j
- Beispiel für n = 3:

1	2	3
2	4	6
3	6	9

- Das die Aufgabe lösende Java-Programm ist zu vervollständigen

```
int n = In.readInt(); if (n < 0) n = -n;                    // no negative values
for
```

Interaktion 17: Beispiel zur for-Anweisung

Interaktion 17 zeigt die Aufgabenbeschreibung eines Programms, die durch eine Schachtelung von zwei for-Anweisungen zu realisieren ist.

Schleifen können bewirken, dass das Ende der Schleife niemals erreicht wird, d.h. das Programm terminiert nicht. Die Semantik der Schleife und damit des ganzen Programms ist in diesem Fall undefiniert, was dem weiter oben eingeführten $\perp$-Symbol entspricht. Ein praktisches Vorgehen, die Terminierung sicher zu stellen, besteht in der Nutzung von Zusicherungen (*Assertions*), die bereits im Zusammenhang mit den Verzweigungen eingeführt wurden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
// The following program section should calculate sum = 1 + 2 + ... + n
i = 1;  sum = 0;
while (i <= n) {
  // Assertion when entering the loop body: i <= n
  sum = sum + i;
  i = i + 1;
}
// Assertion after leaving the loop: i > n
```

- Zwei triviale Zusicherungen (analog Verzweigungen)
  - am Anfang des Schleifenrumpfes
  - nach Verlassen des Schleife

- Eine interessante weitere Zusicherung ist die Schleifeninvariante
  - Aussage über das Ergebnis der Schleife, die in jedem Durchlauf gültig bleibt

Information 18: Zusicherung bei Schleifen

Die in Information 18 enthaltenen zwei Zusicherungen betreffen zum einen Aussagen zu der Abbruchbedingung der Schleife. Eine weitere Zusicherung, die Schleifeninvariante, ist sehr viel schwieriger zu ermitteln und bedarf einiger Erfahrungen.

Am Beispiel einer einfachen Summierungsschleife soll das Prinzip der Schleifeninvariante illustriert werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)


```
i = 1;  sum = 0;
while (i <= n) {
  // sum == add(1 .. i - 1)
  sum = sum + i;
  // sum == add(1 .. i)
  i = i + 1;
  // sum == add(1 .. i - 1)
}
// sum == add(1 .. n)
```

- Schleifeninvariante sum = add(1 .. i - 1)
  - add(1 .. x) berechnet die Summe 1 + ... + x

- Es ist zu zeigen, dass die Schleifeninvariante durch den Schleifenrumpf erhalten bleibt

Interaktion 19: Schleifeninvariante zu einem Beispiel-Programm

Die Frage, was die in Interaktion 19 angegebene Schleife berechnet, lässt sich unmittelbar beantworten. Im Falle einer bereits nur wenig komplexeren Schleife wäre die Antwort und damit die Schleifeninvariante bereits sehr viel schwieriger zu ermitteln.

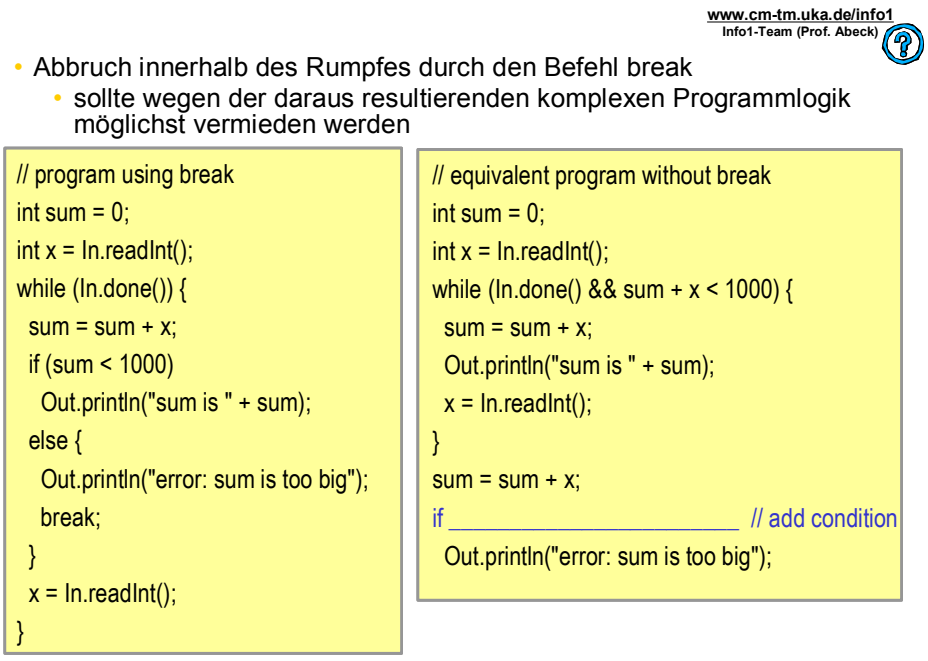
Nach der Festlegung der Schleifeninvariante sum == add(1 .. i - 1) ist nachfolgend zu beweisen, dass es sich auch tatsächlich um eine Invariante handelt. Hierzu zeigt man, dass nach einem Durchlauf durch die Schleife die Invariante erhalten bleibt.

Konkret müssen zum Nachweis der Schleifeninvariante die zwei Anweisungen, die den Schleifenrumpf bilden, untersucht werden.

Die Zusicherung nach Verlassen der Schleife (sum = add(1 .. n)) wird durch eine (Und-) Verknüpfung der Schleifeninvariante und der Zusicherung erreicht, die in Information 18 im Zusammenhang mit der Schleifenbedingung aufgestellt wurde.

Die Terminierung der Schleife lässt sich ebenfalls leicht nachweisen, da zum einen mit n eine obere Schranke gegeben ist und zum anderen der Wert von i, der gegen diese obere Schranke geprüft wird, in jedem Schleifendurchlauf um 1 erhöht wird.

Bei allen drei Schleifenformen (while, do-while, for) wird durch die in Java angebotene break-Anweisung die Möglichkeit zugelassen, nicht über die Schleifenbedingung, sondern über eine im Rumpf durchgeführte Abprüfung einer Bedingung die Schleife zu verlassen.



Interaktion 20: Abbruch von Schleifen

In dem in Interaktion 20 gezeigten linken Programmausschnitt, der die break-Anweisung beinhaltet, ist die Bedingung für den Abbruch der Schleife dann gegeben, wenn die Variable sum nicht kleiner als 1000 ist.

Zum Verständnis des Programms ist ein Blick in die Klasse In und der von ihr bereitgestellten Methode done() hilfreich. Im Java-Quellcode findet sich die folgende Erklärung:

/** Check if the previous operation was successful.

```
This method returns true if the previous read operation was able
to read a token of the requested structure. It can also be called
after open() and close() to check if these operations were successful.
If done() is called before any other operation it yields true.
*/
```

Ein Blick auf die Erklärung der Methode readInt() verschafft dann die endgültige Klarheit:

```
/** Read an integer.
This method skips white space and tries to read an integer. If the
text does not contain an integer or if the number is too big, the
value 0 is returned and the subsequent call of done() yields false.
An integer is a sequence of digits, possibly preceeded by '-'.
*/
```

Die Schleife wird also verlassen, sobald der Benutzer keine korrekte *Integer*-Zahl eingibt.

Wie die gleichwertige Programmvariante auf der rechten Seite zeigt, lässt sich die Verwendung der break-Anweisung in manchen Fällen dadurch umgehen, dass die Abbruchbedingung in der Schleifenbedingung untergebracht wird. Das Programm wird dadurch meist übersichtlicher, da die Zusicherung nach der Schleife klarer formuliert werden kann.

Im Beispiel ist nach dem Durchlauf durch die Schleife in einer if-Anweisung zu ermitteln, ob die Schleife "korrekt" (d.h. ln.done() == false) oder aufgrund der zur Schleifenbedingung hinzugefügten Abbruchbedingung (d.h. (sum + x < 1000) == false) beendet wurde. Die entsprechende Bedingung ist in Interaktion 20 entsprechend zu ergänzen.

3 METHODEN

Methoden bieten eine wichtige Grundlage zur Strukturierung von Programmen. Nachfolgend werden die in den PROGRAMMIERGRUNDLAGEN gemachten Ausführungen um einige fortgeschrittene Konzepte zu den Methoden ergänzt [Mö03].

Zu einem Methodennamen können durch das so genannte Überladen mehrere Methodenausführungen, bestehend aus der formalen Parameterliste und dem Methodenrumpf, in einem Block existieren.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Es darf ein Methodenname mehrfach in einem Block auftreten, wenn sich diese Methoden in ihrer Parameterliste unterscheiden
- Das damit verknüpfte Konzept wird als Überladen von Methoden bezeichnet
  - mit einem Methodennamen sind verschiedene Bedeutungen verbunden
- Es wird diejenige Methode beim Aufruf ausgewählt, deren formale Parameterliste zu der aktuellen Parameterliste am besten passt

Information 21: METHODEN – Überladen von Methoden

Die Auswahl der jeweiligen Methodenausführung erfolgt beim Aufruf aufgrund der aktuellen Parameterliste, wie am nachfolgenden Beispiel verdeutlicht wird.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
static void print(int i) {...}
static void print(float f) {...}              // overload ok: different type
static void print(int i, int width) {...}     // overload ok: different number of parameters

print(100);                                   // calls print(int i)
print(3.14f);                                 // calls print(float f)
print(100, 5);                                // calls print(int i, int width)
```

- Kriterien zur Unterscheidung der Parameterliste
  - Typ der formalen Parameter
  - Anzahl der formalen Parameter

- Die zur Ausgabe verwendete Methode Out.print() ist ein weiteres Beispiel einer überladenen Methode

Information 22: Beispiele für das Überladen von Methoden

Die in Information 22 angegebenen Aufruf-Beispiele ermöglichen eine eindeutige Zuordnung zu einer der drei Varianten der überladenen Methode print(). Die Typangabe müsste hierbei gar nicht zwingend exakt übereinstimmen, da z.B. der Aufruf der print()-Methode mit einer Variablen vom Typ short aufgrund der zulässigen Konvertierung nach int mit der ersten Variante

void print (int i) {...}

ausgeführt wird.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Methoden können zur Zerlegung von Problemen genutzt werden

- Beispiel: Methode zum Kürzen eines Bruchs nutzt die Methode zur Ermittlung des ggT

```
static int gcd (int x, int y) {               // compute greatest common divisor of x and y
  int rest = x % y;                           // modulo operation (rest of x / y)
  while (rest != 0) {
    _________; _________; _________;          // add three missing statements
  }
  return y;
}

static void reduce (int numerator, int denominator) {
  int x = ____________________________; // add right side of assignment
  Out.print((numerator / x) + "/" + (denominator / x));
}
```

Interaktion 23: Problemzerlegung mittels Methoden

Am Beispiel des in Interaktion 23 aufgezeigten Kürzen von Brüchen soll verdeutlicht werden, dass Methoden dazu geeignet sind, die aus der Problemzerlegung resultierenden Teilprobleme wiederum in Form von Methoden zu lösen. In diesem Fall besteht das Problem aus dem Kürzen eines Bruchs. Ein hierin auftretendes Teilproblem ist die Ermittlung des größten gemeinsamen Teilers (ggT, *Greatest Common Divisor* GCD), das durch eine separate Methode gelöst wird.

Die zum Kürzen erforderliche Ermittlung des größten gemeinsamen Teilers erfordert den bereits kennen gelernten und bekannten Euklidischen Algorithmus, der in der Methode gcd() in Interaktion 23 realisiert ist und in der zweiten Methode reduce(), die das Kürzen eines übergebenen Bruchs realisiert, genutzt wird.

Falls die Methode gcd() bereits bestehen sollte (z.B. innerhalb einer Bibliothek mathematischer Funktionen), würde diese zur Lösung des Problems "Kürzen von Brüchen" wieder verwendet werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Wiederverwendung von Code
  - Programm wird kürzer und lesbarer
  - Nutzung von Bibliotheken verkürzt die Erstellungszeit und macht die Programme robuster
- Erweiterung des Angebots an Operationen
  - die von einer Sprache angebotenen Grundoperationen lassen sich durch eigene Operationen erweitern
  - Durchbrechen der Begrenzungen der Sprache
- Strukturierung von Programmen
  - Aufteilen einer unstrukturierten langen Anweisungsfolge in überschaubare und logisch zusammen gehörige Programmstücke
  - gezielte Verwendung von sprechenden Methodennamen ermöglicht ein Verständnis auch ohne die genaue Analyse der einzelnen Anweisungen

Information 24: Gründe für die Nutzung von Methoden

Die drei in Information 24 genannten Gründe, die für den Einsatz von Methoden sprechen, treffen alle für das eben behandelte Beispiel zu.

4 DATENSTRUKTUREN

Neben den ablauforientierten Sprachelementen werden in der Programmierung auch datenorientierte Sprachelemente benötigt, durch die Einzelwerte in Datenstrukturen zusammen geführt werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Die zunächst behandelten Datenstrukturen, die intensiv in der imperativen Programmierung genutzt werden, sind Arrays und Strings

- Array (Feld)
  - eine ein- oder mehrdimensionale Tabelle von Elementen
  - Elemente haben alle den gleichen Typ
  - Zugriff auf die Array-Werte über Nummern (Index)

- String (Zeichenkette)
  - vergleichbar einem Array von Zeichen
  - aufgrund der hohen Bedeutung dieser Datenstruktur wird in vielen Sprachen (wie auch in Java) ein eigener Typ eingeführt
  - Zahlreiche Operationen (z.B. Vergleiche)

Information 25: DATENSTRUKTUREN - Überblick

Mit den Arrays und den Strings werden im Folgenden die zwei elementaren Datenstrukturen behandelt, die in jeder modernen imperativen Programmiersprache angeboten werden.

4.1 Arrays

Die Datenstruktur der Arrays bietet die Möglichkeit, Variablen gleichen Typs zu einer Wertemenge zusammen zu fassen.

4.1.1 Arbeiten mit eindimensionalen Arrays

Ein Array kann aus mehreren Dimensionen bestehen. Zunächst wird die einfachste Form, das eindimensionale Array, das einem Vektor von Werten entspricht, näher betrachtet.

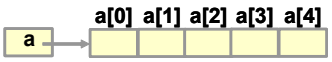
www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Zunächst Betrachtung von eindimensionalen Arrays

- Wichtigste Eigenschaften eines Arrays
  - hat einen Namen
  - ermöglicht den Zugriff auf Array-Elemente über einen Index
    - alle Elemente sind vom gleichen Typ
    - Elemente sind Speicherzellen und verhalten sich wie namenlose Variablen

- Beispiel:

```
int[] a;              // Declaration of an array of int elements
a = new int[5];       // Generation of an array with 5 int elements
```



Information 26: Arrays

Wie Information 26 zeigt, erfolgt der Zugriff auf die einzelnen Elemente des Arrays über einen Index. Die Deklaration eines Arrays erkennt man an den eckigen Klammern [], die an einen beliebigen Typ angehängt werden. Im Beispiel wird mit

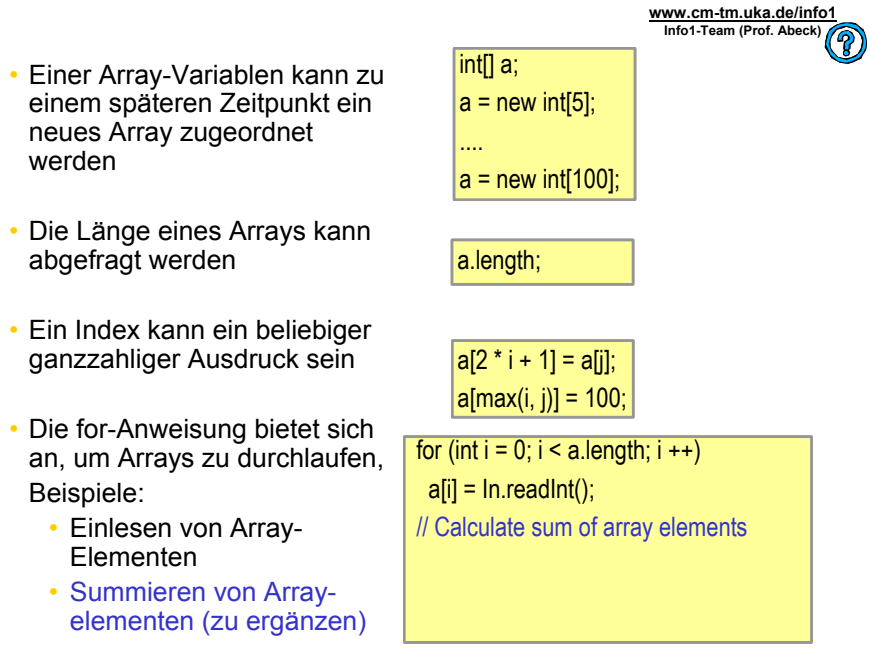
```
int[] a;
```

ein Array aus int-Elementen mit der Bezeichnung a deklariert.

Die eigentliche Speicherreservierung wird durch den new-Operator erbracht, der im Beispiel

```
a = new int[5];
```

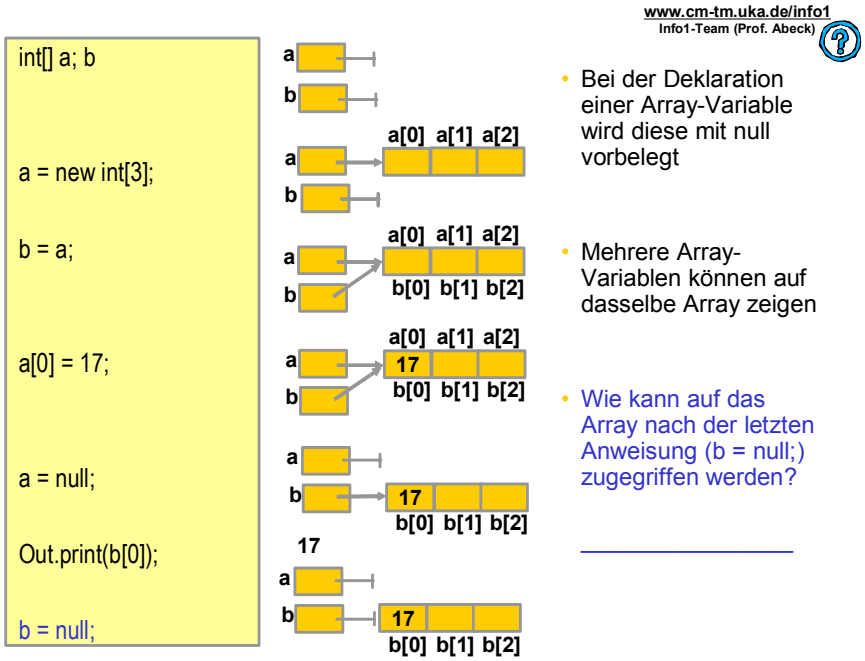
Speicher für fünf int-Elemente vorsieht, die über die Array-Variable a unter Angabe eines Indexes zugegriffen werden können.



Interaktion 27: Arbeiten mit Arrays

Wie Interaktion 27 verdeutlicht, kann einer Array-Variablen, der bereits ein erzeugtes Array zugewiesen wurde, zu einem späteren Zeitpunkt ein anderes Array zugewiesen werden. Dieses Array muss zu dem in der Deklaration angegebenen Typ kompatibel sein und kann kürzer oder länger als das zuvor zugewiesene Array sein. Auf Werte, die eventuell in dem zuvor zugewiesenen Array eingetragen wurden, lässt sich nach einer Zuweisung eines neuen Arrays nicht mehr über diese Array-Variable zugreifen.

Wie das in Interaktion 27 entsprechend zu ergänzende Beispiel einer Array-Bearbeitung zeigt, bietet sich die for-Anweisung zum Durchlaufen der Array-Elemente an.



Interaktion 28: Array-Zuweisung

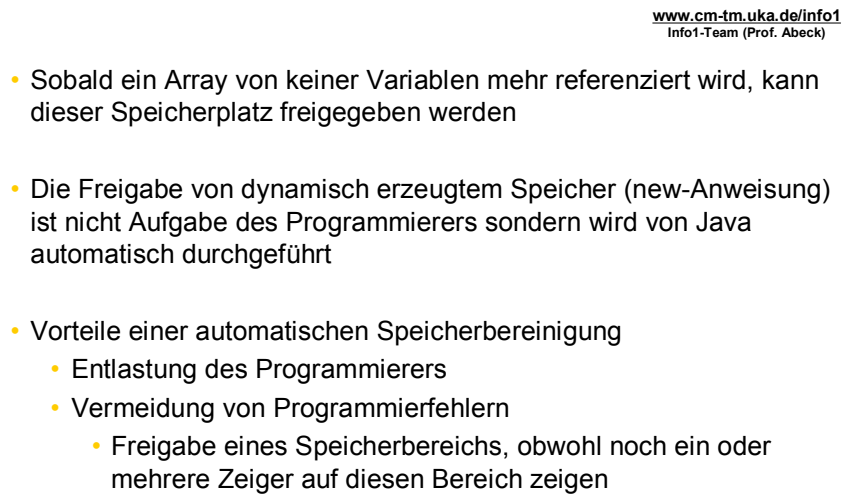
Anhand eines konkreten Ablaufs sollen die verschiedenen Formen der Array-Zuweisung verdeutlicht werden (siehe Interaktion 28). Zunächst werden zwei Array-Variablen `a` und `b` deklariert. Anschließend wird `a` ein mittels `new` erzeugtes 3-elementiges Array zugeordnet. Durch die Array-Zuweisung

```
b = a;
```

zeigt `b` auf dasselbe Array wie `a`. Hierdurch kann auch dann noch auf die Werte dieses Arrays zugegriffen werden, selbst wenn `a` ein anderes Array (im Beispiel das `null`-Array) zugewiesen werden sollte. `null` ist der so genannte *Nullpointer*, was bedeutet, dass der Array-Variablen kein Array zugeordnet ist.

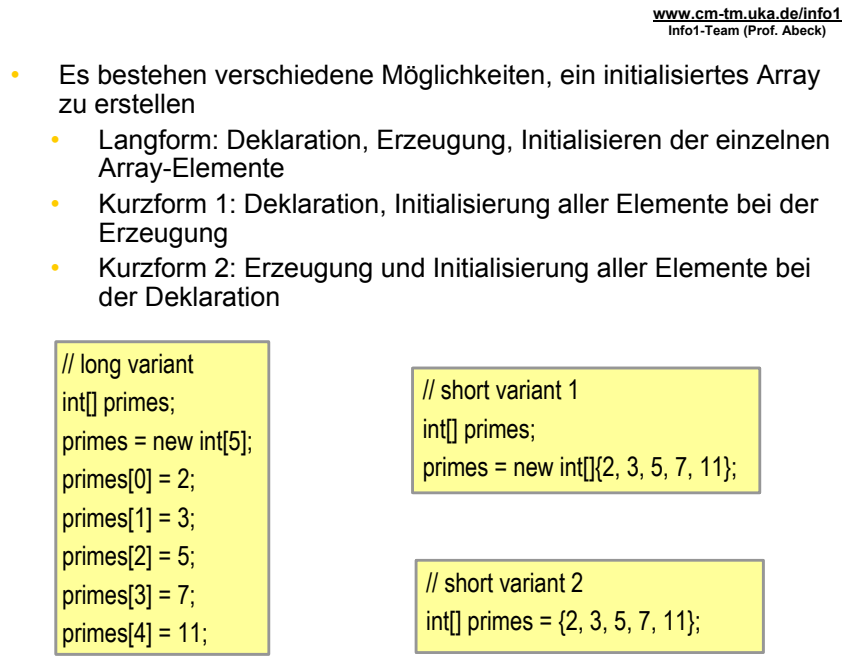
4.1.2 Freigabe von Arrays

Nach Durchführung der obigen Anweisungsfolge zeigt keine Array-Variable auf den Array-Speicher, womit dieser Speicher nicht mehr zugänglich ist und damit frei gegeben werden kann.



Information 29: Speicherbereinigung (*Garbage Collection*)

Wie in Information 29 ausgeführt ist, erfolgt die Freigabe und Bereinigung des Speichers, die *Garbage Collection*, automatisch durch das Java-Laufzeitsystem, was erhebliche Vorteile für den Programmierer mit sich bringt. In Sprachen wie z.B. C bedeutet die durch den Programmierer selbst durchzuführende Speicherfreigabe einen erheblichen Aufwand und eine Quelle schwer zu behebender Programmierfehler.



Information 30: Varianten zur Erstellung von initialisierten Arrays

Am Beispiel eines 5-elementigen Primzahlen-Arrays wird in Information 30 verdeutlicht, dass Java elegante und die Lesbarkeit erhöhende Möglichkeiten anbietet, die Initialisierung eines Arrays mit der Erzeugung (Kurzform 1) oder mit der Deklaration und der Erzeugung in einer Anweisung zu verbinden.

4.1.3 Suchen in Arrays

Arrays werden zur Lösung zahlreicher Programmierprobleme benötigt. Ein typisches solches Beispiel ist das Suchen eines Eintrags in einem Katalog (z.B. Telefonbuch, Warenbestand, Kurskatalog).

- Eingabe von seqSearch()
  - gesuchter Zahlwert
  - Array mit ganzen Zahlen, in dem der Zahlenwert gesucht wird
- Ausgabe von seqSearch()
  - Position des gesuchten Zahlwerts im Array, falls der Zahlwert im Array enthalten ist
  - -1, falls Zahlwert nicht im Array enthalten ist

```
/* searches elem in a[] sequentially
 * returns position in a[], if found and -1, if not found
*/
static int seqSearch (int elem, int[] a) {
  int pos = a.length - 1;                    // pos initialized to position of last array element
  while (pos >= 0 && a[pos] != elem)
    pos--;
  // afterloop assertion: _____________________________________
  return pos;
}
```

Interaktion 31: Sequentielle Suche

Das Beispiel in Interaktion 31 zeigt eine einfache Such-Funktion, die eine Zahl x in einem Array int[] a von ganzen Zahlen sucht, indem die Zahlen im Array von der letzten Position pos beginnend mit dem gesuchten Zahlenwert verglichen werden.

Die angegebene Lösung liefert zwar das gewünschte Ergebnis, ist aber offensichtlich nicht effizient. Insbesondere bei großen Katalogen (z.B. mit 10^4 Einträgen und mehr) stellt sich das sequentielle Durchlaufen des Arrays als nicht akzeptabel – weil zu aufwändig – heraus.

Durch eine Anforderung an die Eingabe lässt sich die Effizienz der Suche erheblich steigern, wie in Information 32 näher ausgeführt wird.

- Analoge Beschreibung der Such-Funktion mit folgender neuer Anforderung an die Eingabe:
  - das Array, in dem der Zahlenwert gesucht werden soll, liegt sortiert vor

```
/* searches elem in a[] by binary intersection of search interval
 * param first, last define search interval
 * returns position in a[], if found and -1, if not found
*/
static int binSearch (int elem, int[] a, int first, int last) {
  if (first > last) return -1;        // search interval empty; elem not found
  int m = (first + last) / 2;         // select middle position m in search interval
  if (elem == a[m]) return m;    // found
  if (elem < a[m]) return binSearch(elem, a, first, m-1);         // search elem in first half
  return binSearch(elem, a, m+1, last);   // only reached if elem > a[m]
}
```

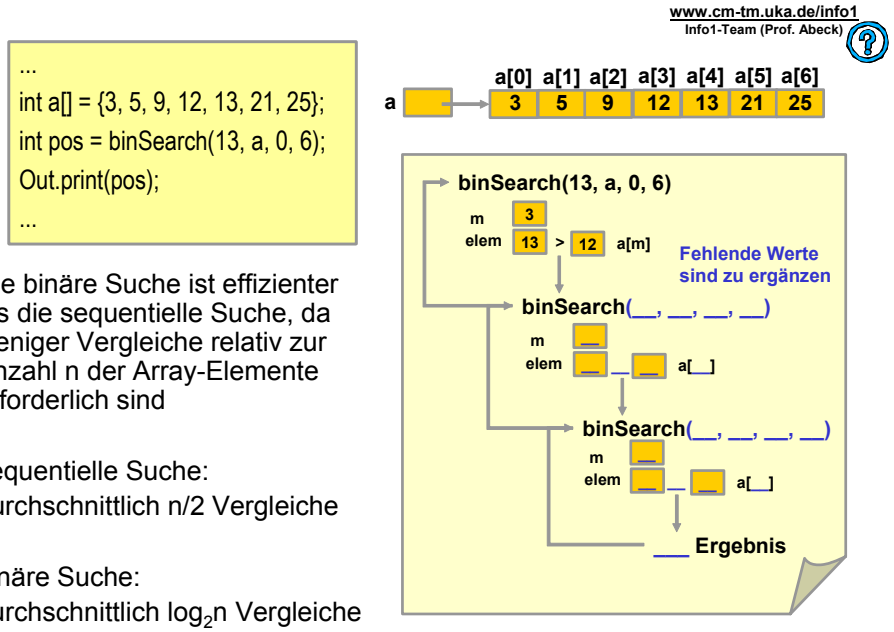
Information 32: Binäre Suche

Die Anforderung besteht darin, dass das Array a[], in dem der Zahlenwert elem gesucht wird, sortiert ist. In diesem Fall kann anstelle der sequentiellen Suche eine so genannte binäre Suche erfolgen, die den Namen aufgrund der fortlaufenden Zweiteilung (binäre Zerlegung) des Arrays erhält.

Es bietet sich an, die binäre Suche durch eine Methode binSearch() zu realisieren, wie in Information 32 angegeben ist. Diese Methode hat die interessante Eigenschaft, dass sie sich selbst aufruft, weshalb sie als rekursive Methode bezeichnet wird. Auf Rekursionen wird in der Kurseinheit RECHENSTRUKTUREN UND FUNKTIONALE PROGRAMME genauer eingegangen.

Aufgrund der Sortiert-Eigenschaft des Arrays lässt sich nach Auswahl eines Elements – im Algorithmus wird das bezüglich der Position in der Mitte (int m = (first + last) / 2) des Arrays liegende Element gewählt – entscheiden, ob das gewählte Element das gesuchte Element ist oder falls nicht, ob das Element in der ersten oder zweiten Hälfte des Arrays zu suchen ist.

Das auf diese Weise entsprechend eingegrenzte Suchintervall erfolgt durch die Angabe der Position des ersten (first) und des letzten (last) Elements im Array als Parameter der Methode binSearch().



- Die binäre Suche ist effizienter als die sequentielle Suche, da weniger Vergleiche relativ zur Anzahl n der Array-Elemente erforderlich sind
- sequentielle Suche: durchschnittlich n/2 Vergleiche
- binäre Suche: durchschnittlich $\log_2 n$ Vergleiche

Interaktion 33: Beispielablauf zur binären Suche

Die Arbeitsweise des Algorithmus, der der binären Suche zugrunde liegt, wird am Beispiel des in Interaktion 33 skizzierten Beispielablaufs verdeutlicht.

Da bei der binären Suche jeder Vergleich bewirkt, dass die Anzahl der noch zu durchsuchenden Elemente im Durchschnitt halbiert wird, ist dieses Vorgehen im Vergleich zur sequentiellen Suche sehr viel effizienter. Ist n die Anzahl der Elemente des Arrays, so ist der Aufwand des Algorithmus, der die sequentielle Suche ausführt, von linearer Ordnung, also O(n), während der Algorithmus zur binären Suche nur einen Aufwand logarithmischer Ordnung, also O(log n) bedeutet.

Es sei daran erinnert, dass die effiziente binäre Suche nur dann erfolgen kann, wenn das Array sortiert ist. Sortieralgorithmen stellen aus diesem Grund eine wichtige Algorithmenklasse in der Informatik dar.

4.1.4 Zeichen-Arrays

Array-Elemente können von einem beliebigen Typ sein, so z.B. auch Zeichen, was zu den so genannten Zeichen-Arrays oder char-Arrays führt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Deklaration von Zeichen-Arrays analog zu bisher behandelten Zahlen-Arrays
- In einem neu erzeugten Array enthalten alle Elemente den Wert \u0000
- Kurzformen zur Deklaration und Erzeugung bzw. zur Deklaration, Erzeugung und Initialisierung

```
char[] s;   // declaration
            // no array generated so far

s = new char[20];
            // 20 array elements generated
            // and assigned to s

char[] s1 = new char[20];
            // declaration and generation

char[] s2 = {'a', 'b', 'c'};
            // declaration, generation
            // and initialization
```

Information 34: Zeichen-Arrays

Im Grundsatz lassen sich die auf die Zahlen-Arrays bezogenen Ausführungen zur Deklaration, zur Erzeugung und zur Initialisierung auch auf Zeichen-Arrays (oder auch Arrays eines anderen Typs) anwenden.

Solange die Elemente in einem erzeugten Zeichen-Array nicht initialisiert wurden, weist Java diesen das voreingestellte (Default) Unicode-Zeichen \u0000 zu.

Auch in Zeichen-Arrays ist wie in Zahlen-Arrays die Suche eine nahe liegende Problemstellung. Die in Interaktion 35 gestellte Aufgabe besteht darin, das Auftreten einer Zeichenkette pat (steht für *Pattern*, d.h. Muster) in einer Zeichenkette t zu finden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Eingabe
  - zwei Zeichen-Arrays t und pat
- Ausgabe
  - Position des ersten Auftretens von pat in t, falls pat in t vorkommt
  - -1 sonst

```
static int stringPos (char[] t, char[] pat) {
  int i, j;
  int last = t.length - pat.length;          // last possible position of pat in t
  for (i=0; i <= last; i++) {
    if (t[i] == pat[0]) {                    // first character of pat matches
    j = 1;
    while (j < pat.length && pat[j] == t[i+j]) j++;
    _____________________                    // found
    } // if
  } // for
    _____________________                    // not found
}
```

Interaktion 35: Suche in einem Zeichen-Array

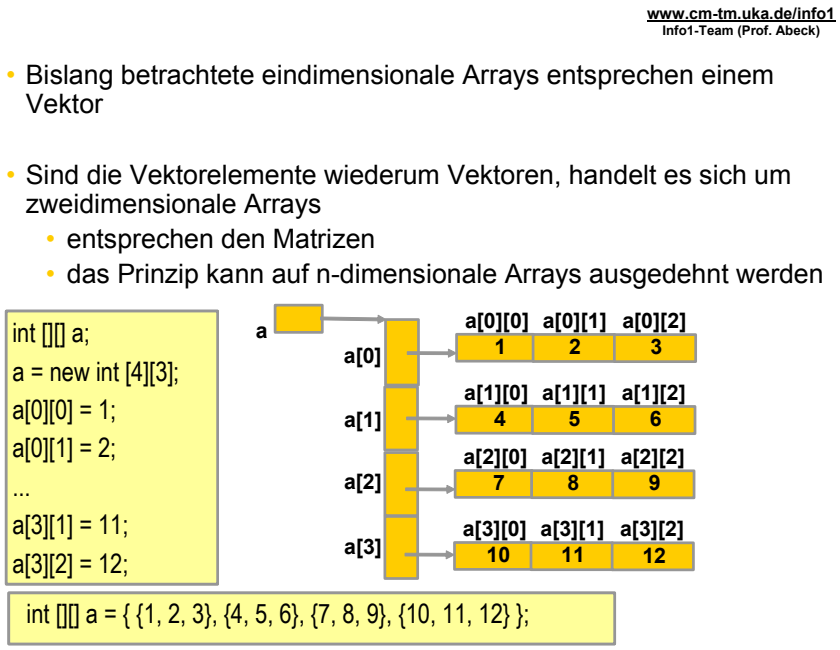
Der Algorithmus zu diesem Problem liegt auf der Hand: Bis zu der letzten möglichen Position last, in der pat in t auftreten kann, wird in einer äußeren for-Schleife überprüft, ob das erste Zeichen von pat (also pat[0]) mit dem untersuchten (i+1)-tenZeichen in t (also t[i]) übereinstimmt.

Wird ein solches erstes übereinstimmendes Zeichen gefunden, wird in der inneren while-Schleife dann festgestellt, ob die nachfolgenden Zeichen in t ebenfalls mit den weiteren Zeichen in pat übereinstimmen. Kann eine Übereinstimmung für alle Zeichen in pat festgestellt werden ist die gesuchte Position gefunden. Hierbei ist zu beachten, dass pat[length(pat)-1] das letzte Zeichen in pat ist.

In Interaktion 35 ist das Java-Programm um die entsprechenden Anweisungen zu ergänzen, die insbesondere die Ergebnisrückgabe der Methode stringPos() realisieren.

4.1.5 Mehrdimensionale Arrays

Arrays können als Elemente wiederum Arrays enthalten, was zu den mehrdimensionalen Arrays führt.



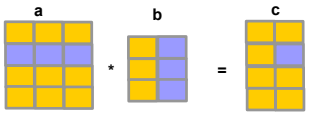
Information 36: Mehrdimensionale Arrays

Die beiden in Information 36 gezeigten Programmausschnitte sind gleichbedeutend: Ein zweidimensionales Array mit 4 Zeilen und 3 Spalten, also eine 4x3-Matrix, wird nach dessen Deklaration (int a[][]) und Erzeugung (a = new int [4][3]) mit den Werten 1 bis 12 (zeilenweise) initialisiert. Wie bereits bei den eindimensionalen Arrays kennen gelernt, lässt sich dieser Vorgang in einer einzigen Anweisung zusammenfassen.

Der Umgang mit mehrdimensionalen Arrays soll am Beispiel der Matrixmultiplikation verdeutlicht werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Eingabe
  - Matrizen a[n][y] und b[y, m]
- Ausgabe
  - Matrixprodukt c[n,m] von a und b



c[1][1] = a[1][0]*b[0][1] + a[1][1]*b[1][1] + a[1][2]*b[2][1]

```
static float[][] matMult (float [][] a, float [][] b) {
  // assertion: _____________________________________________
  float [][] c = new float[a.length][b[0].length];
  int i, j, k;
  for (i = 0; i < a.length; i++)              // for all rows of a
    for (j = 0; j < b[0].length; j++) {       // for all columns of b
      float sum = 0;
      for (k = 0; k < b.length; k++) sum = sum + a[i][k] * b[k][j];
    c[i][j] = sum;
    } // for
  return c;
}
```

Interaktion 37: Matrixmultiplikation

Die Multiplikation von zwei Matrizen a und b setzt voraus, dass die Spaltenanzahl der Matrix a mit der Zeilenanzahl der Matrix b übereinstimmt. In Interaktion 37 ist das Prinzip der Matrixmultiplikation für eine 4x3-Matrix a und eine 3x2-Matrix b gezeigt, deren Matrixprodukt a * b eine 2x4-Matrix c ergibt.

Die Methode matMult() erhält die zwei zu multiplizierenden Matrizen als Eingabe und liefert das Matrixprodukt als Ergebnis. Die oben beschriebene Anforderung an die zu multiplizierenden Matrizen ist als eine Zusicherung im Programm zu ergänzen.

4.2 Strings

Im vorhergehenden Abschnitt wurden mit den Zeichen-Arrays bereits eine mit den Strings vergleichbare Art einer Datenstruktur vorgestellt. Da diese Datenstruktur, die als Zeichenketten bezeichnet werden, sehr häufig auftritt, wurde in Java ein eigener Bibliothekstyp String vorgesehen.

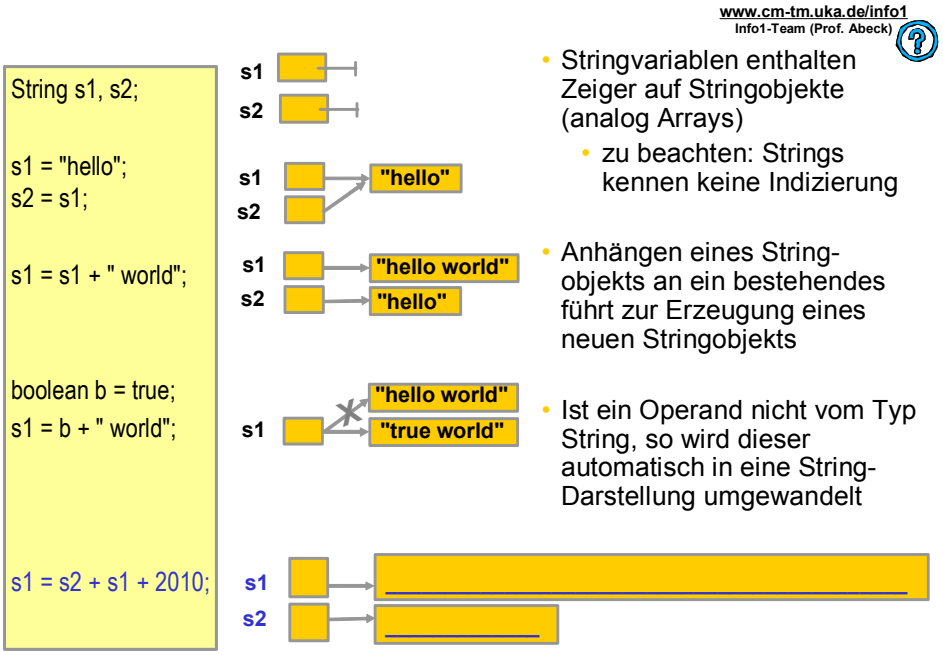
www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- String ist ein von Java bereit gestellter Bibliothekstyp
  - teilweise in die Sprache integriert, weil der Compiler in bestimmten Fällen speziellen Code für Stringoperationen erzeugt
- Stringkonstanten sind Zeichenfolgen, die in doppelte Hochkommata gestellt sind
  - z.B. "a string", "containing a \" character", "\u03c0 is pi"
- Unterschied "x" und 'x'
  - "x" ist eine Stringkonstante mit nur einem Zeichen
  - 'x' ist eine Zeichenkonstante vom Typ char

Information 38: Strings

String ist nicht wie die bislang kennen gelernten Datentypen (z.B. int, float, char) ein in der Sprache Java verankerter Typ, sondern ein Bibliothekstyp. Da der Compiler speziellen Code für Stringoperationen erzeugen kann, besteht aber ein enger Bezug mit der Sprache Java.

Stringkonstanten sind an den doppelten Hochkommata zu erkennen und enthalten eine endliche Menge von Zeichen. Es können die zur Darstellung von Zeichen bestehende *Escape*-Sequenzen in der in Information 38 gezeigten Art und Weise in Stringkonstanten genutzt werden. Stringkonstanten bestehen aus beliebig vielen Zeichen; sie können also auch nur aus einem Zeichen oder auch aus keinem Zeichen ("") bestehen, was als leerer String bezeichnet wird.

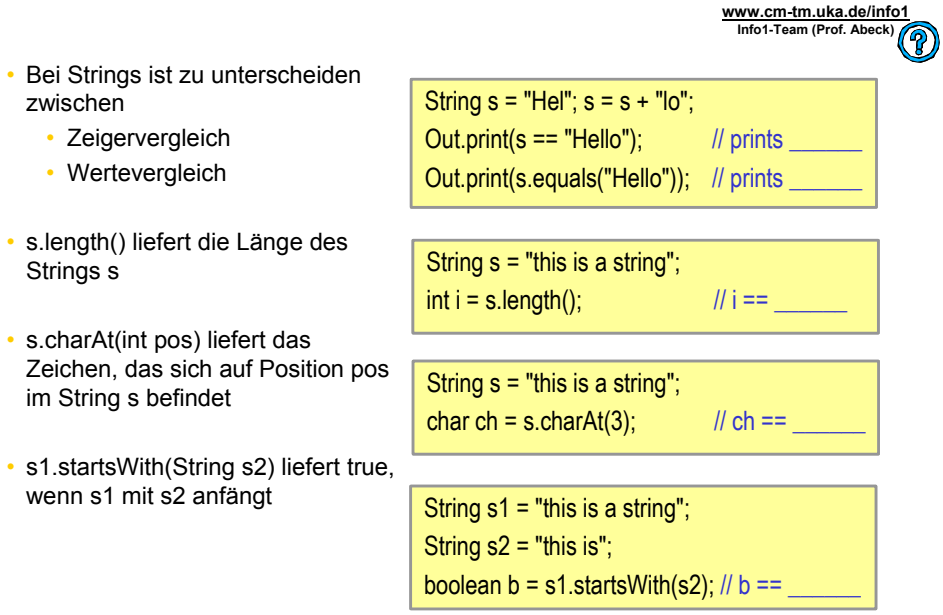


Interaktion 39: Stringvariablen

Strings weisen eine zu den Arrays ähnliche Zeiger-Charakteristik auf. Durch den Operator + lassen sich Strings verketten. Hieran wird deutlich, dass ein Compiler den String-Typ kennen muss, obwohl dieser nicht in der Sprache, sondern nur in der Bibliothek definiert ist.

4.2.1 Stringvergleich

Im Zusammenhang mit Strings sind zwei Arten von Vergleichen zu unterscheiden. Die Operation == führt einen Vergleich auf der Zeigerebene durch – d.h. sie liefert dann true, wenn die miteinander verglichenen Strings auf dasselbe Stringobjekt zeigen. Sollen die Strings daraufhin verglichen werden, ob sie die gleichen Zeichenfolgen als Werte haben, ist nicht ein Zeiger-Vergleich, sondern ein Werte-Vergleich erforderlich. Hierzu stellt die Bibliothek für den Stringtyp eine geeignete Methode equals() zur Verfügung.



Interaktion 40: Stringvergleich und weitere Stringoperationen

Der Werte-Vergleich equals() ist nur eine von zahlreichen Operationen, die im Zusammenhang mit Strings dem Java-Programmierer zur Bearbeitung von Strings von der Bibliothek angeboten werden. Interaktion 40 beschreibt eine kleine Auswahl von Methoden, deren Aufruf jeweils in einem Programmausschnitt verdeutlicht wird.

4.2.2 Stringmanipulationen

Da das einer Stringvariablen zugewiesene Objekt konstant ist, eignen sich diese Variablen nicht für dynamische Veränderungen, durch die das Objekt schrittweise manipuliert werden kann.

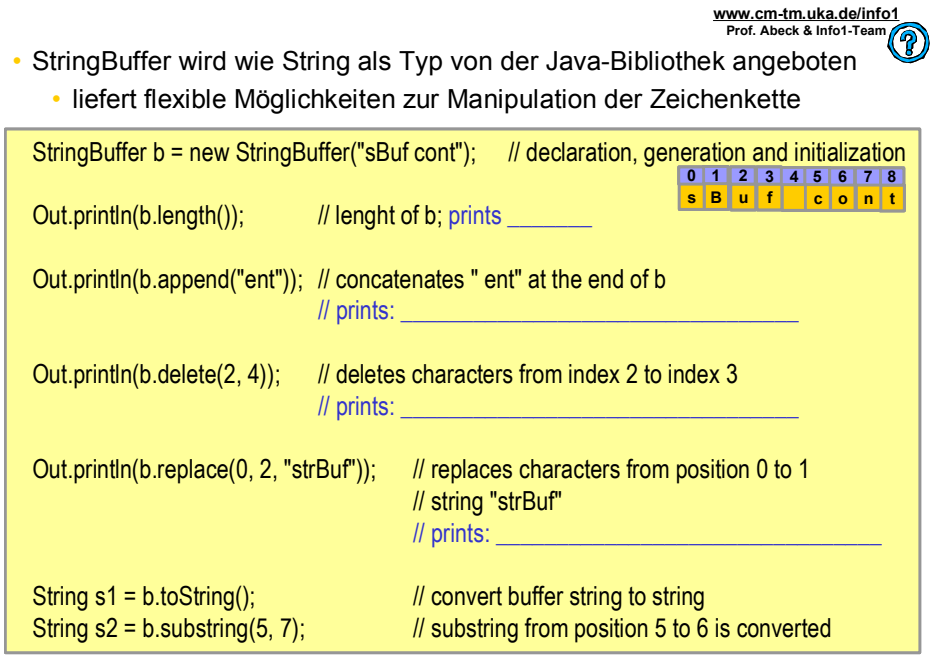
- Stringobjekte sind konstant und eignen sich nicht dazu, flexible Stringmanipulationen anzubieten
- Zwei Möglichkeiten: char-Arrays oder StringBuffer
- char-Arrays
  - Zusammensetzen des Strings im char-Array und anschließende Umwandlung in einen String

```
char[] a = new char[80];
for (int i = 0; i < 80; i++) a[i] = In.read();
String s1 = new String(a);              // generation of string object
String s2 = new String(a, 0, 40);       // first 40 characters a[0] ... a[39]
```

Information 41: Stringmanipulationen

Information 41 zeigt mit den char-Arrays eine von zwei Möglichkeiten auf, solche Manipulationen durchzuführen. In dem Beispiel wird die Zeichenkette in einer Schleife zeichenweise eingelesen. Anschließend wird ein Stringobjekt s1 erzeugt, dem der Inhalt des

gesamten char-Arrays als Wert zugewiesen wird. Wie anhand der Erzeugung des zweiten Stringobjekts s2 ersichtlich, kann durch die Angabe zweier Positionsangaben nur ein Teil eines char-Arrays in ein Stringobjekt umgewandelt werden.



Interaktion 42: StringBuffer

Der Typ StringBuffer wird wie der Typ String durch die Java-Bibliothek zur Verfügung gestellt. Ein Objekt dieses Typs verhält sich im Wesentlichen wie ein String und bietet zusätzlich umfangreiche Operationen zur Manipulation des zugewiesenen Stringwertes. An dem in Interaktion 42 gezeigten Programmausschnitt werden einige dieser vom Typ StringBuffer bereitgestellten Operationen an einem konkreten Beispiel verdeutlicht.

Die von StringBuffer bereitgestellten Methoden, die Manipulationen an dem im Puffer gehaltenen Wert vornehmen, sind Funktionen, die jeweils den veränderten Pufferinhalt zurückgeben.

Die im Beispiel benutzten Methoden stellen nur eine kleine Auswahl dar. Es sei angemerkt, dass in vielen Methoden neben Strings auch andere Typen als Parameter zulässig sind. So erlaubt beispielsweise die Methode append(x) mittels Überladung, dass x vom Typ char, int, long, float, double, boolean, String oder char[] ist.

4.2.3 Stringkonversion

Die Konversion, also die Umwandlung von Strings in andere Typen und umgekehrt, ist eine häufig in einem Programm auftretende Aufgabe. Daher stellt die Java-Bibliothek für diesen Zweck geeignete Funktionen zur Verfügung.

- Die Java-Bibliothek bietet geeignete Funktionen, um den Wert eines bestimmten Typs in einen String umzuwandeln und umgekehrt

```
String s = String.valueOf(x);          // converts value of x to a string
                                       // type of x can be char, int, long, float, double,
                                       // boolean or char[]

int i = Integer.parseInt("123");       // converts an integer-like char sequence to its
                                       // corresponding integer value

float f = Float.parseFloat("3.14");    // converts a float-like char sequence to its
                                       // corresponding float value

char[] a = s.toCharArray();            // provides char array containing value of string s
```

Information 43: Stringkonversionen

Zur Konversion in einen String stellt der String-Typ die Methode valueOf() bereit. Aufgrund der angenehmen Eigenschaft, dass der Operator + (String-Konkatenation) Werte automatisch in Strings konveriert, muss diese Methode nicht so häufig genutzt werden.

Die Umwandlung eines Strings in einen Wert eines bestimmten anderen Typs erfolgt gemäß der in Information 43 aufgeführten Methoden. Bei der Umwandlung in die Typen int bzw. float ist zu beachten, dass der an die Methoden parseInt() bzw. parseFloat() übergebene String auch tatsächlich eine zulässige "*Integer*-Zeichenkette" bzw. "*Float*-Zeichenkette" beinhaltet. Andernfalls liefert das Java-Laufzeitsystem eine so genannte Ausnahme (*Exception*).

Ausnahmen und deren Behandlung werden in der Kurseinheit FORTGESCHRITTENE PROGRAMMIERKONZEPTE behandelt.
Mit den in dieser Kurseinheit eingeführten Anweisungen und Datenstrukturen lassen sich aufbauend auf den in der Kurseinheit PROGRAMMIERGRUNDLAGEN beschriebenen Sprachelemente bereits größere imperative Java-Programme entwickeln.

VERZEICHNISSE

Abkürzungen und Glossar

Abkürzung oder Begriff	Langbezeichnung und/oder Begriffserklärung
⊥	*Bottom*-Element Symbolisiert im Zusammenhang mit Zuständen den gedachten „Endzustand“ nicht-terminierender Programme. Synonymer Begriff: Undefiniert-Element
Array	Eine Datenstruktur, durch die Variablen gleichen Typs zu einer Wertemenge zusammengefasst werden können.
BNF	Backus-Naur-Form Schreibweise für Regeln einer Grammatik
EBNF	Erweiterte BNF Erweiterung der BNF um Metaregeln zur ausdrucksstärkeren Formulierung von Regeln einer Grammatik.
ENV	*Environment* Menge der Werte-Belegung von Identifikatoren, die durch Variablen dargestellt werden.
Hornerschema	Schema zur effizienten Berechnung von Polynomen.
Methode	Aus der objektorientierten Programmierung stammende Bezeichnung für eine Rechenvorschrift, die aus einem Methodennamen und formalen Parametern besteht. Handelt es sich bei der Methode um eine Funktion, so ist zusätzlich der Typ des von der Methode zurückgegebenen Ergebnisses anzugeben. Methoden sind in Java das Mittel, mit dem Botschaften realisiert werden [MS+03].
Nullpointer	Zeiger (*Pointer*), dessen Wert undefiniert ist, da ihm noch keine Speicheradresse zugewiesen wurde.
O(n)	lineare Ordnung Schreibweise (O-Notation), durch die der Aufwand eines Algorithmus ausgedrückt wird.
Rekursive Methode	Eine Methode, die sich in ihrem Methodenrumpf selbst aufruft.
UML	*Unified Modeling Language* Sprache, die mittels graphischer Elemente eine semi-formalen Beschreibung von beliebigen Gegenständen (z.B. Software-Systeme oder Geschäftsbereiche) ermöglicht.
WWW, Web	World Wide Web Standardisierte Anwendung des Internet, durch die u.a. ein

Kommunkationsprotokoll (*Hypertext Transfer Protocol* HTTP) und eine Sprache zur Beschreibung von Web-Seiten (*Hypertext Markup Language* HTML) festgelegt werden.

Zusicherung — Explizite Aussage über den Programmzustand in Form eines booleschen Ausdrucks über Werte der im Programm genutzten Größen.
Englischer Begriff: *Assertion*

Zuweisung — Anweisung, durch die einer Variablen ein Wert zugeordnet wird.
Englischer Begriff: *Assignment*

Index

Informationen und Interaktionen

Literatur

[Br98] Manfred Broy: Informatik – Eine grundlegende Einführung, Band 1: Programmierung und Rechenstrukturen, Springer Verlag 1998.

[Mö03] Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

[MS+03] Stefan Middendorf, Reiner Singer, Jörn Heid: Java – Programmierhandbuch für die Java-2-Plattform, Standard Edition, dpunkt.verlag, 2003.

OBJEKTORIENTIERTE PROGRAMMIERUNG

Kurzbeschreibung

Das der objektorientierten Programmierung zugrunde liegende Klassenkonzept sowie wichtige Klassen dynamischer Datenstrukturen und das Vererbungsprinzip werden behandelt.

Schlüsselwörter

Klasse, Geheimnisprinzip, Konstruktor, dynamischer Datentyp, Liste, Vererbung, Polymorphie, dynamische Bindung, abstrakte Klasse, *Framework*, *Interface*, Schnittstellenklasse

Lernziele

1. Das Klassenkonzept als Basis der Objektorientierung wird konzeptionell und praktisch durchdrungen.
2. Das auf dem Klassenkonzept aufsetzende Vererbungsprinzip und die dynamische Bindung werden verstanden.
3. Programme, die das Klassenkonzept und weiterführende objektorientierte Prinzipien nutzen, können geschrieben werden.

Hauptquellen

- Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

Inhaltsverzeichnis

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- KLASSEN
  - Deklaration und Verwendung, Klassen und Arrays, Array-Stack
- OBJEKTORIENTIERUNG
  - Konstruktoren, Klassenelemente und Objektelemente
- DYNAMISCHE DATENSTRUKTUREN
  - Listen (unsortiert und sortiert), Listen-Stack
- VERERBUNG
  - Polymorphie, Dynamische Bindung, abstrakte Klassen, Interfaces, Frameworks

Information 1: OBJEKTORIENTIERTE PROGRAMMIERUNG

1 KLASSEN

Die objektorientierte Programmierung bildet die Basis der heute in der Praxis verwendeten Softwaretechnik [Mö03].

Die der Objektorientierung zugrunde liegenden Techniken setzen auf den in der Kurseinheit IMPERATIVE PROGRAMMIERUNG beschriebenen Konzepten auf. Zu den zentralen Konzepten der Objektorientierung gehören die Klassen und die daraus hervorgehenden Objekte. Die Objekte sind dabei Gegenstände der realen oder der künstlichen Welt, wie in der Kurseinheit GRUNDBEGRIFFE DER INFORMATIK im Zusammenhang mit der Modellierung dargestellt wurde.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Klassen sind selbst definierte Datentypen, durch die mehrere Elemente zusammengefasst werden können
- Die Elemente können aus verschiedenartigen Elementen aufgebaut sein
  - wesentlicher Unterschied zu Arrays
- Beispiel
  - es soll eine Klasse aufgebaut werden, die ein Datum beschreibt
  - Elemente sollen sein
    - Tag beschrieben als ganzzahliger Wert
    - Monat beschrieben als Zeichenkette
    - Jahr beschrieben als ganzzahliger Wert

Information 2: KLASSEN -
Wesentliche Eigenschaften und Beispiel

Durch Klassen werden verschiedene bestehende Datentypen zu einem neuen, übergeordneten Datentyp zusammengefasst. Hinsichtlich dieses Aspekts haben Klassen eine Ähnlichkeit mit den in der Kurseinheit IMPERATIVE PROGRAMMIERUNG eingeführten Arrays. Ein wesentlicher

Unterschied zu Arrays ist, dass Klassen aus verschiedenartigen Elementen aufgebaut sein können.

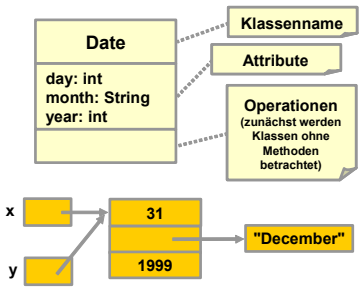
1.1 Beispielklasse Date

Wie im Beispiel in Information 2 ausgeführt ist, kann eine Datums-Klasse beispielsweise aus zwei ganzzahligen Werten, die den Tag und das Jahr beschreiben, sowie einer den Monat angebenden Zeichenkette zusammengesetzt sein.

```
class Date {                        // class declaration
  int day;
  String month;
  int year;
}

class DateProgram {
  public static void main (String[] arg) {
    Date x, y;            // variable declaration
    x = new Date();   // object generation
    x.day = 31; x.month = "December"; x.year = 1999;
    y = x;                // assignment of object variable
    Out.println(y.day + " " + y.month + " " + y.year);
  } // main
} // class
```

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

UML-Klassendiagramm



- Deklaration durch das Schlüsselwort class
- Objekterzeugung durch das Schlüsselwort new
- Zuweisung von Objektvariablen wie bei Arrays
  - bewirkt, dass der Zeiger auf das zugewiesene Objekt verweist

Information 3: Deklaration und Verwendung von Klassen

Das Schlüsselwort class wurde bereits früher im Zusammenhang mit der Grundstruktur eines Java-Programms eingeführt (siehe Kurseinheit PROGRAMMIERGRUNDLAGEN [C&M-PG]).

Wie das UML-Klassendiagramm in Information 3 verdeutlicht, werden zunächst Klassen betrachtet, die nur Attribute und keine Methoden besitzen. Die deklarierte Klasse Date besitzt die drei zuvor eingeführten Attribute day, month und year.

Wie bereits im Zusammenhang mit den Arrays kennen gelernt, werden durch die Deklaration Date x, y lediglich Objektvariablen angelegt, die auf kein Objekt zeigen, also den Nullpointer null als Wert beinhalten. Die Erzeugung eines Date-Objekts und dessen Zuweisung zur Objektvariablen x erfolgt durch x = new Date(). Nach der Belegung dieses Objekts mit einem konkreten Datum (31 December 1999) erfolgt mittels y = x die Zuweisung des Wertes von x zur Objektvariablen y und der Zugriff über y auf die Attribute des Objekts im Rahmen des Methodenaufrufs Out.println(y.day + " " + y.month + " " + y.year).

Hinsichtlich der Zuweisung von Objektvariablen stellt sich unmittelbar die Frage, welche Eigenschaften die Objektvariablen erfüllen müssen, um typkompatibel zu sein.

1.2 Typkompatibilität und Vergleich von Objekten

Im Folgenden werden Zuweisungen und Vergleiche von Objekten näher betrachtet.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- In Java ist eine Zuweisung von Objektvariablen nur zulässig, wenn die Objektvariablen den gleichen Typnamen haben
  - diese Art der Typkompatibilität heißt Namensäquivalenz
- Eine andere Art von Typkompatibilität ist die Strukturäquivalenz
  - zwei Typen sind gleich, wenn sie die selbe Struktur haben
  - ist in Java wegen geforderter Namensäquivalenz nicht zulässig
- Namensäquivalenz lässt sich durch einen Compiler einfacher überprüfen als Strukturäquivalenz

Information 4: Typkompatibilität

Die Sprache Java unterstützt hierbei mit der in Information 4 beschriebenen Namensäquivalenz eine im Vergleich zur Strukturäquivalenz restriktivere Typkompatibilität.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Folgende zwei Arten von Vergleichen lassen sich bei Objekten unterscheiden

(1) Vergleich auf der Ebene der Objektvariablen (Zeigervergleich)

```
Date x, y;  if (x==y) ....; if (x!=y) ...
```

(2) Vergleich auf der Ebene der Attribute

```
static boolean isDateEqual (Date x, Date y) {
  return x.day == y.day && x.month.equals(y.month) && x.year == y.year;
}
...
if (isDateEqual(x,y)) ...
```

- Richtig oder falsch?

	richtig	falsch
Aus der Objektgleichheit folgt die Attributgleichheit	☐	☐
Aus der Attributgleichheit folgt die Objektgleichheit	☐	☐

Interaktion 5: Objektgleichheit und Attributgleichheit

Wie Interaktion 5 verdeutlicht, können Objekte auf der Objektvariablen-Ebene und damit auf der Zeigerebene oder auch auf der Attribut-Ebene miteinander verglichen werden. Offensichtlich haben zwei Objektvariablen, die auf dasselbe Objekt zeigen (also x == y) auch übereinstimmende Attributwerte (also isDateEqual(x, y)). Übereinstimmende Attributwerte lassen aber keine Aussage darüber zu, ob es sich um dasselbe oder zwei verschiedene Objekte handelt.

1.3 Objekte als Rückgabewerte

Bislang wurden ausschließlich als Methoden realisierte Funktionen betrachtet, die nur einen einfachen Wert zurückgeliefert haben. Da der Rückgabewert auch ein einzelnes Objekt sein kann, lassen sich hierüber beliebige, in dem Objekt zusammengefasste Ergebniswerte durch eine Methode zurückliefern.

- Eine Methode (Funktion) kann ein Objekt als Ergebnis zurückliefern
  - hierdurch kann eine Funktion mehrere Werte zurückgeben

```
class Time { int h, m, s;}                  // hours, minutes, seconds

class ConvertTimeProgram {
  static Time convert (int seconds) {   // convert seconds to time
    Time t = new Time();                // generate time object
    t.h = seconds / 3600;
    t.m = (seconds % 3600) / 60;        // rest from (seconds / 3600) divided by seconds per minute
    t.s = seconds % 60;
    return t;
  }

  public static void main (String[] arg) {
    int seconds; Time t;
    Out.print("enter seconds: "); seconds = In.readInt();
    while (In.done()) {
      t = convert(seconds); Out.println(t.h + ":" + t.m + ":" + t.s);
     Out.print("enter seconds: "); seconds = In.readInt();
} /* while */    } /* main */    } /* class */
```

Information 6: Objekt als Rückgabewert einer Methode

Am Beispiel der im Java-Programm von Information 6 enthaltenen Methode convert() zu der Klasse ConvertTimeProgram wird deutlich, dass innerhalb dieser Methode drei int-Werte (t.h, t.m. t.s) berechnet werden, die mittels return t in einem Time-Objekt zusammengefasst zurückgeliefert werden.

1.4 Klassen und Arrays

Die vorhergehenden Ausführungen haben gezeigt, dass zwischen Klassen und Arrays gewisse Zusammenhänge bestehen, die im Folgenden näher untersucht werden.

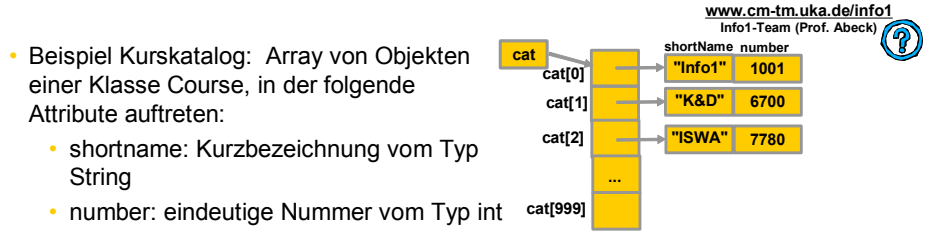
- Gemeinsamkeiten von Klassen und Arrays
  - bestehen aus mehreren Elementen
  - werden durch Zeiger referenziert

- Unterschiede
  - Arrays bestehen aus gleichartigen Elementen, während Klassen aus verschiedenartigen Elementen bestehen
  - Array-Elemente haben keinen Namen, Klassen-Elemente (Attribute) haben einen Namen
  - Anzahl der Elemente wird zu unterschiedlichen Zeitpunkten festgelegt
    - Array: bei der Erzeugung des Array-Objekts
    - Klasse: bei der Deklaration

Information 7: Klassen und Arrays

Information 7 stellt die Gemeinsamkeiten und Unterschiede der beiden Datenstrukturen im Überblick dar. Die Unterschiedlichkeit der Elemente von Arrays und Klassen im Hinblick auf Art, Name und Festlegung der Anzahl verdeutlicht, dass die beiden Datenstrukturen trotz der offensichtlichen Gemeinsamkeiten ganz unterschiedliche Zielsetzungen verfolgen.

Häufig werden beide Datenstrukturen kombiniert in einem Programm verwendet, wie am Beispiel der Programmierung eines Kurskatalogs in Interaktion 8 verdeutlicht wird.




```
class Course {String shortName; int number;}      // declaration of class Course

class CourseCatProgram {
  static Course[] cat;                            // catalog as array of courses
  static int courseCount = 0;                     // counts the number of courses in the catalog

  public static void main (String[] arg) {
    int i; cat = new Course[1000];                // generation of array of 1000 pointers to course objects
    cat[courseCount] = new Course();              // generation of one new course object
    cat[courseCount].shortName = "Info1"; cat[courseCount].number = 1001; courseCount++;
    // add courses describing K&D and ISWA
    for (i = 0; i < courseCount; i++) Out.println(cat[i].shortName + "   " + cat[i].number);
  } /* main */    } /* class */
```

Interaktion 8: Kombination von Klassen und Arrays

Unter einem Kurs wird hierbei eine Lehrveranstaltung, wie z.B. die INFORMATIK-I-Veranstaltung verstanden. Im Beispiel sind zwei weitere Kurse K&D (KOMMUNIKATION UND DATENHALTUNG [C&M-KuD]) und ISWA (INTERNET-SYSTEME UND WEB-APPLIKATIONEN [C&M-ISWA]) genannt.

Ein Kurs wird im Beispiel durch die Klasse Course mit den zwei Attributen shortName (Kurzbezeichnung) und number (eindeutige Kursnummer) beschrieben. Es wären zahlreiche weitere Attribute denkbar, wie z.B. der den Kurs anbietende Dozent, die Art der Veranstaltung (z.B. Vorlesung oder Praktikum) oder die Anzahl der Semesterwochenstunden, die der Kurs umfasst.
Der Kurskatalog cat wird durch ein Array von Kursen Course[] implementiert, wobei insgesamt maximal 1000 Kurseinträge (new Course [1000]) im Katalog aufgenommen werden können.

Durch die nachfolgende Anweisungsfolge in der main()-Methode werden die drei oben beschriebenen Kurse erfasst und in einer for-Anweisung ausgegeben.

Im obigen Java-Programm wurden Informationen nur für den ersten Kurs zugewiesen. Das Programm ist entsprechend zu ergänzen, damit der Katalog mit dem im Diagramm in Interaktion 8 gezeigten Zustand übereinstimmt.

Offensichtlich bieten sich für die Erfassung und die Ausgabe, aber auch für andere Katalog-Operationen (z.B. Suche oder das Löschen) separate Methoden an. Diese Überlegungen führen zu der im folgenden Kapitel näher ausgeführten Objektorientierung.

2 OBJEKTORIENTIERUNG

Bislang lag der Schwerpunkt auf dem Datenaspekt von Klassen. Dabei wurde deutlich, dass Klassen dazu geeignet sind, semantisch zusammengehörige Daten in eine Datenstruktur zusammen zu führen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Durch Klassen lassen sich nicht nur die Daten, sondern auch die dazugehörigen Methoden zu einer Einheit zusammenfassen
  - diese Einheit von Daten und darauf operierende Methoden schaffen eine geeignete Ordnung im Programm

- Beispiel: Klasse zur Beschreibung eines Kurses
  - Daten: Kurzbezeichnung, Nummer
  - Methoden
    - Erfassen der Kursdaten
    - Ändern der Kursdaten
    - Abfrage der Kursdaten

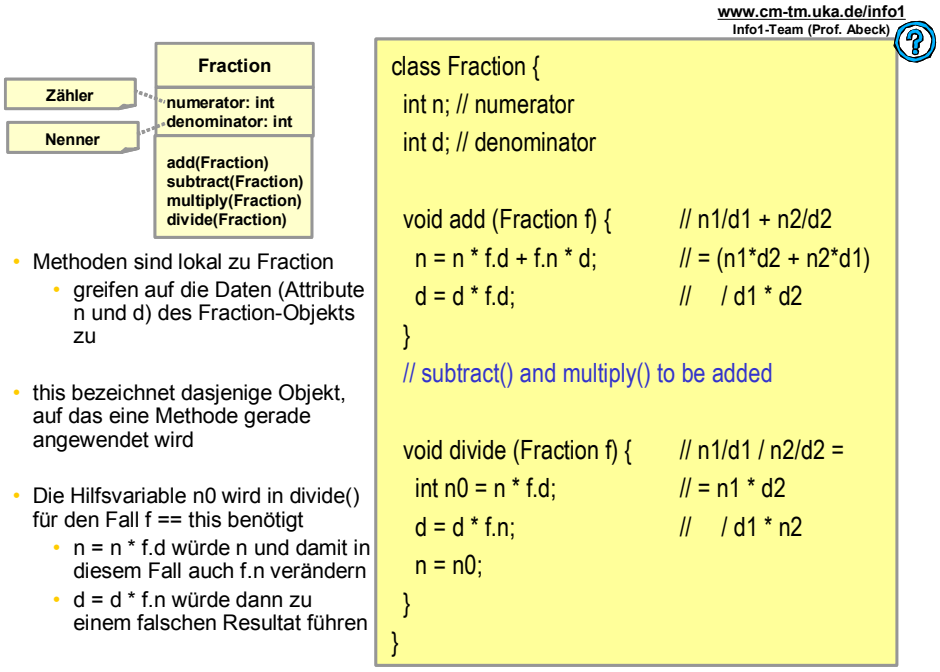
Information 9: OBJEKTORIENTIERUNG - Daten und Methoden als eine Einheit

Wie in Information 9 ausgeführt ist, lassen sich mittels Klassen nicht nur Daten, sondern insbesondere Daten und die darauf arbeitenden Operationen, also die Methoden zusammenfassen. Die Einheit aus Daten und Methoden stellt eine geeignete Form zur Gliederung komplexer Software-Systeme dar.

Programme, die beispielsweise die im letzten Abschnitt betrachtete Klasse Course nutzen möchten, stehen damit neben den Daten gleichzeitig auch alle auf diesen Daten sinnvoll anwendbaren Methoden zur Verfügung.

2.1 Beispiel: Klasse Fraction

Am Beispiel des Bruchrechnens und der Klasse Fraction wird die Zusammenfassung von Daten und Methoden als zentrales Konzept der Objektorientierung verdeutlicht.



Interaktion 10: Beispiel-Klasse Fraction

Interaktion 10 zeigt die Klasse Fraction mit den Daten numerator und denominator sowie den vier Methoden add(), subtract(), multiply() und divide() als graphische UML-Beschreibung und als unvollständige Java-Klasse, die um die fehlenden Methoden zu ergänzen ist.

Wie am Beispiel der Methode add() zu erkennen ist, kann im Rumpf einer Methode ein Zugriff auf die in der Klasse lokal deklarierten Daten erfolgen. Diese Daten sind dem Objekt zugeordnet, dessen Methoden aufgerufen werden. Java bietet mit dem Schlüsselwort this die Möglichkeit, dieses Objekt explizit anzusprechen.

Am Beispiel der Methode add() soll der Umgang mit dem Objekt this verdeutlicht werden. In der ersten Zeile

```
n = n * f.d + f.n * d;
```

bedeuten n und d den Zugriff auf die Daten des Objekts, auf das die Methode angewendet wird, weshalb die Zuweisung auch gleichbedeutend lauten könnte:

```
this.n = this.n * f.d + f.n * this.d;
```

Es ist nicht ausgeschlossen, dass eine Methode mit dem Objekt als formalen Parameter aufgerufen wird, auf das die Methode angewendet wird. Aus diesem Grund muss in der Methode divide() eine Methoden-lokale Variable zur Aufnahme des Zwischenresultats eingeführt werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
Fraction a = new Fraction(); a.n = 1; a.d = 2;
Fraction b = new Fraction(); b.n = 3; b.d = 5;

a.add(b);          // a = 1/2 + 3/5 = 11/10
b.divide(b);       // b = 3/5 * 3/5 = 15/15
```

- Objektorientierte Sprechweise zum Aufruf der Methode a.add(b):
  - Objekt a bekommt die Meldung (oder den Auftrag) add
    - hierdurch Aufruf der Methode add()
  - Objekt a, das die Meldung erhält, wird der Empfänger der Meldung genannt

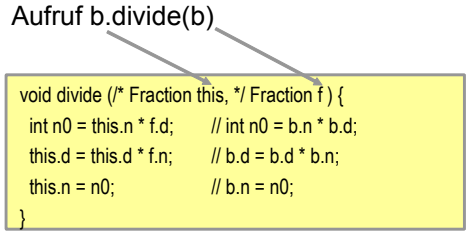
- Die Fraction-Methoden haben neben dem formalen Parameter f noch einen zweiten (versteckten) formalen Parameter this
  - beim Aufruf a.add(b) wird this der aktuelle Parameter a und f der aktuelle Parameter b zugewiesen
  - zu beachten: dem formalen Parameter f kann als aktueller Parameter auch das Empfänger-Objekt zugeordnet werden

Information 11: Methodenaufruf

In Information 9 werden zwei Objekte a und b der Klasse Fraction erzeugt und die Klassenvariablen werden so initialisiert, dass a den Bruch 1/2 und b den Bruch 3/5 beinhaltet.

In den nachfolgenden Anweisungen erhält der Empfänger a die Meldung add mit b als aktuellen Parameter und b erhält die Meldung divide mit dem aktuellen Parameter b. In der erstgenannten Anweisung wird der Standardvariablen this beim Aufruf der Methode der Wert a und in der zweiten Anweisung der Wert b zugewiesen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)



- Warum wird die Variable n0 benötigt?

_____

- Könnte man die Variable statt n0 auch n nennen?

_____

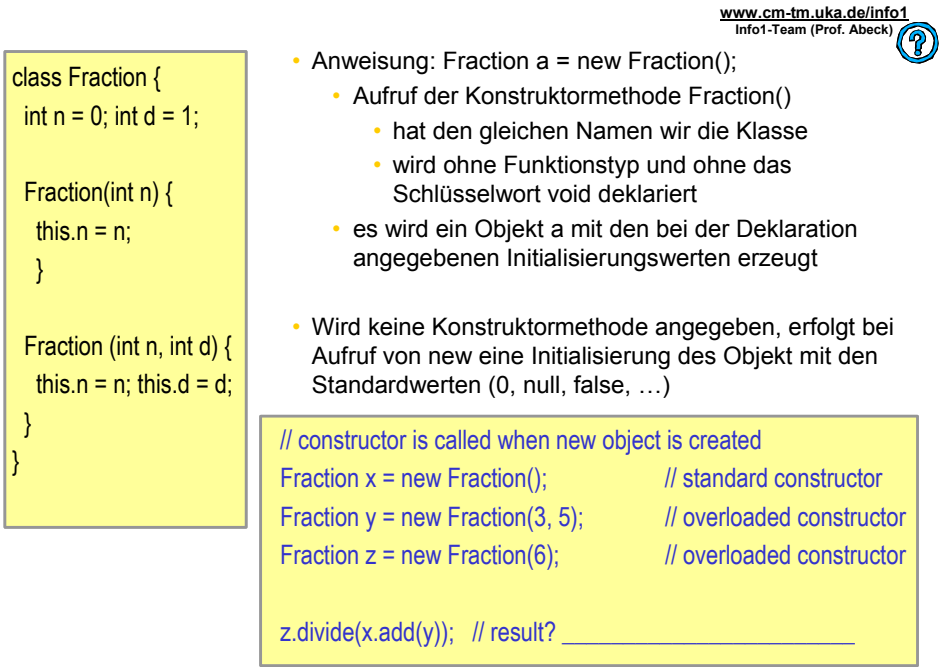
Interaktion 12: Versteckter Parameter this im Methodenaufruf

Interaktion 12 führt den versteckten Parameter this beim Methodenaufruf durch entsprechende Kommentare im Methodenkopf explizit auf. Am Beispiel des konkreten Aufrufs b.divide(b) kann man sich klarmachen, warum die Einführung einer lokalen Hilfsvariablen n0 erforderlich ist.

Die zweite Frage zur Benennung dieser Variablen zielt darauf ab, ob hierdurch im Falle einer Umbennenung zu n ggf. ein Konflikt mit dem gleichnamigen Attribut n entsteht.

2.2 Konstruktoren

Konstruktoren sind spezielle Methoden, die bei der Erzeugung eines Objekts automatisch aufgerufen werden und die dazu genutzt werden können, dieses Objekt in der gewünschten Form zu initialisieren.



Interaktion 13: Konstruktormethoden

Konstruktormethoden haben im Vergleich zu den bislang kennen gelernten Methoden einige besondere Eigenschaften. Neben den in Interaktion 13 genannten Namens- und Typ-Charakteristika ist zu beachten, dass ein Konstruktor niemals explizit, sondern immer nur implizit während der Erzeugung eines Objekts (durch das Schlüsselwort new) aufgerufen wird.

Die bereits beschriebene Eigenschaft des Methodenüberladens kann dazu genutzt werden, eigene Konstruktoren einzuführen. Die in Konstruktoren erlaubten formalen Parameter werden dabei üblicherweise dazu genutzt, die Initialisierungswerte der Klassenvariablen bei der Objekterzeugung mittels new zu übergeben, wie das Beispiel in Interaktion 13 verdeutlicht. Es wird zudem ersichtlich, dass mehrere Konstruktoren bestehen können.

Die Angabe eines oder mehrerer Konstruktoren zu einer Klasse ist optional. Java fügt bei fehlendem Konstruktor automatisch einen parameterlosen Standard-Konstruktor hinzu, der die Attribute mit den gemäß ihres Typs bestehenden Standardwerten (0, null, false, ...) belegt.

Eine andere Form der Initialisierung besteht darin, den zu initialisierenden Wert bei der Deklaration der Klassenattribute anzugeben. Falls keine eigenen Konstruktoren existieren, treten diese Initialisierungswerte an die Stelle der Standard-Initialisierungswerte. Im Falle selbst geschriebener Konstruktoren werden die Initialisierungen ggf. nochmals verändert. Bezogen auf das Beispiel in Interaktion 13 bedeutet diese Festlegung, dass beispielsweise das Objekt y nach dessen Erzeugung nicht die Initialisierungswerte der Deklaration, sondern die durch den Konstruktor-Aufruf zugewiesenen Werte beinhaltet.

2.3 Klassenattribute und Klassenmethoden versus Objektattribute und Objektmethoden

In der Beispiel-Klasse Fraction wurden erstmals Methoden verwendet, die nicht mit dem Schlüsselwort static deklariert wurden. Dieser Sachverhalt betrifft die in der Objektorientierung wichtige Unterscheidung zwischen klassenbezogenen (statischen) und objektbezogenen (dynamischen) Attributen und Methoden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Sowohl die Klasse selbst als auch die von dieser Klasse erzeugten Objekte können Attribute und Methoden haben
  - eine Klasse ist selbst ein Objekt, das wie eine Schablone das Aussehen aller Objekte dieser Klasse festlegt
- Syntaktische Unterscheidung
  - Klassenattribute und -methoden werden mit dem Schlüsselwort "static" gekennzeichnet
- Klassenattribute existieren nur einmal pro Klasse, Objektattribute werden getrennt für jedes erzeugte Objekt angelegt

Information 14: Klassen- und objektbezogene Attribute und Methoden

Objektattribute werden erst zur Zeit der Objekterzeugung angelegt und sind diesem speziellen Objekt zugeordnet. Eine Änderung eines Objektattributwertes durch eine Objektmethode hat keine Auswirkung auf den Wert des gleichnamigen Objektattributs eines anderen Objekts.
Im Gegensatz dazu bestehen Klassenattribute nur einmal pro Klasse, weshalb eine Änderung eines Klassenattributs für alle Objekte wirksam wird.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
class Fraction {
  static int fractionCounter;                 // class attribute
  int n = 1; int d = 1;                       // object attributes

  static { fractionCounter = 0;}              // class constructor; only one is allowed
  static void resetFractionCounter () {       // class method
    fractionCounter = 0;
  }

  Fraction () { ....}                         // constructor method(s)
  Fraction (int n, int d) { ...}              // more than one constructor is allowed

  void add (Fraction f) { ... }               // object methods
  ...
}
```

Information 15: Klassen- und objektbezogene Elemente in der Klasse Fraction

Der in Information 15 gezeigte Programmausschnitt zur Klasse Fraction macht deutlich, wie die verschiedenen klassen- und objektbezogenen Elemente in einer Klassendefinition genutzt werden können. Wie das Beispiel zeigt, besteht nicht nur die Möglichkeit, objektbezogene Konstruktormethoden einzuführen. Es kann auch ein Klassenkonstruktor benutzt werden, um die Klassenattribute (hier fractionCounter) zu initialisieren.

Der Zugriff auf Klassenattribute und Klassenmethoden kann im Falle von Mehrdeutigkeiten durch die Angabe des Klassennamens qualifiziert werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- In jedem Java-Programm kommt eine spezielle (öffentliche, also public) Klassenmethode main() vor
- Konvention: diese Klassenmethode kann von der Kommandozeile aus wie ein Programm aufgerufen werden
- Die main()-Methode kann beim Aufruf mit Parametern versehen werden
  - Beispiel: java MyProg test 12
    - hierdurch werden zwei Strings "test" und "12" als Parameter im Stringarray arg übergeben

Information 16: Spezielle Klassenmethode main()

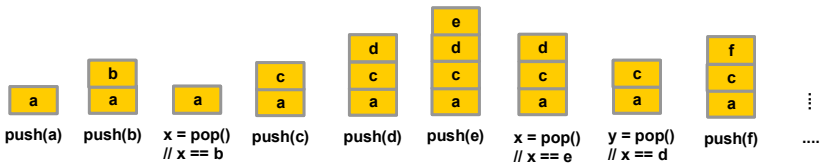
Die Klassenmethode main() spielt in einem Java-Programm eine besondere Rolle, da hierdurch die Einsprungstelle in das Programm vorgegeben wird. Erreicht wird dieses Verhalten durch die in Information 16 angegebene Konvention, die auf der Bezeichnung der Methode basiert.

2.4 Stapel als Beispiel einer Klasse

Der in diesem Abschnitt als eine Klasse beschriebene Stapel – auch Keller bzw. *Stack* genannt – ist eine der elementarsten und wichtigsten Datenstrukturen der Informatik.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Wichtige Datenstruktur der Informatik, durch die ein bestimmtes Zugriffsprinzip auf eine Elementemenge vorgegeben wird
- Zugriffsprinzip: Das zuletzt auf den Stapel gelegte Element wird als erstes wieder entfernt
  - wird als Kellerprinzip oder LIFO-Prinzip bezeichnet
    - Last In First Out
- Stapeloperationen sind
  - push(x) ein Element x auf den Stapel legen
  - x = pop() das oberste Element x vom Stapel nehmen

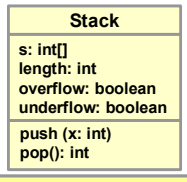


Information 17: Stapel (Keller, *Stack*) als Beispiel einer Klasse

Durch den Stapel wird ein ganz bestimmtes, in Information 17 beschriebenes Zugriffsprinzip vorgegeben, durch das die Wirkungsweise der Schreiboperation push() und der Leseoperation pop() festgelegt ist. Anhand des beispielhaften Verlaufs eines Stapelinhalts lässt sich diese auch als Kellerprinzip oder LIFO-Prinzip (*Last In First Out*) bezeichnete Charakteristik der Datenstruktur einfach nachvollziehen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
class Stack {
  int[] s;    // implementation of stack
              //as an array of integer
  int length;           // current number of
                        // elements in stack
  boolean overflow;
  boolean underflow;

  Stack (int size) {    // constructor
    s = new int[size]; // maximal number of
                        // elements in stack
    length = 0;
    overflow = false;
    underflow = false;
  }
```



UML-Klassendiagramm

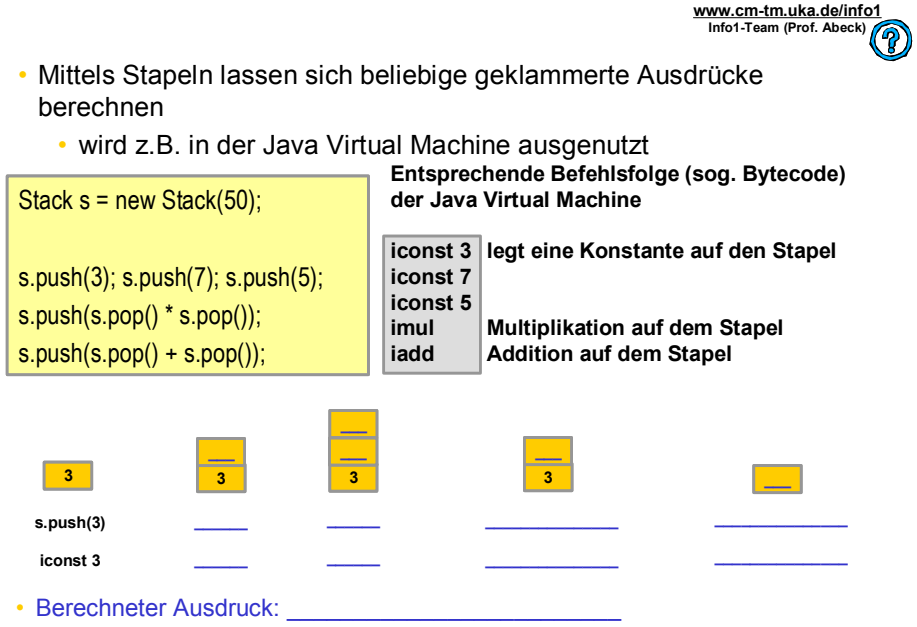
```
  void push (int x) {
    overflow = length >= s.length;
    if (!overflow) s[length++] = x;
  }

  int pop () {
    underflow = length == 0;
    if (!underflow) return s[--length];
    else return -1;
  }
}
```

Information 18: UML-Klassendiagramm und Java-Programm zu Stack

Bei der Umsetzung der Datenstruktur muss u.a. entschieden werden, welche Elemente auf dem Stapel gehalten und wie die Daten gespeichert werden sollen. In der exemplarischen

Implementierung eines Stapels in Information 18 sind die Stapelelemente ganze Zahlen, die in einem int-Array gehalten werden.

In beiden Methoden push() und pop() muss vor dem Schreiben bzw. Lesen des Elements zunächst überprüft werden, ob der Stapel voll (overflow) bzw. leer (underflow) ist. In diesem Fall kann kein Element auf den Stapel geschrieben bzw. vom Stapel gelesen werden.



Interaktion 19: Verwendung von Stapeln

An einem einfachen Anwendungsbeispiel der Klasse Stack kann man sich die grundsätzliche Bedeutung dieser Datenstruktur für die Informatik klarmachen. Stapel lassen sich dazu nutzen, beliebige Ausdrücke zu berechnen, indem die Operanden in gewisser Reihenfolge auf den Stapel geschrieben (push) werden und auf die jeweils obersten Elemente die im Ausdruck auftretenden Operationen ausgeführt werden. Im Rahmen der Ausführung der Operation wird das Operationsergebnis anstelle der Operanden auf den Stapel geschrieben.

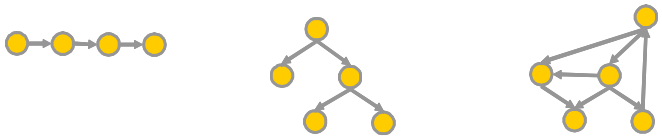
Genau in dieser Art und Weise arbeitet z.B. auch die zum Java-Laufzeitsystem gehörende *Java Virtual Machine* (JVM), wie anhand des Beispiels im Rahmen der Interaktion 19 skizziert werden soll.

3 DYNAMISCHE DATENSTRUKTUREN

Zahlreiche in der Informatik relevante Datenstrukturen sind dadurch gekennzeichnet, dass sie sich dynamisch zur Laufzeit aufbauen und manipulieren lassen. Aufgrund dieser Eigenschaft werden sie als dynamische Datenstrukturen bezeichnet.

- Dynamische Datenstrukturen bestehen aus verketteten Knoten
  - Knoten können dynamisch zur Laufzeit erzeugt und anschließend verkettet werden
  - Knoten können dynamisch zur Laufzeit wieder entfernt oder umgehängt werden

- Typische dynamische Datenstrukturen



Interaktion 20: DYNAMISCHE DATENSTRUKTUREN - Die wichtigsten Arten

Interaktion 19 zeigt drei typische dynamische Datenstrukturen:

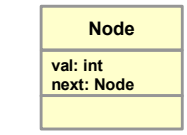
1. **Listen**: mit Ausnahme des letzten Knotens hat jeder Knoten genau einen Zeiger auf den Nachfolger-Knoten.
2. **Baum**: jeder Knoten kann mehrere Zeiger auf Nachfolger-Knoten haben und wird nur durch maximal einen Vorgänger-Knoten referenziert.
3. **Graph**: jeder Knoten kann mehrere Zeiger auf Nachfolger-Knoten und kann durch mehrere Vorgänger-Knoten referenziert werden.

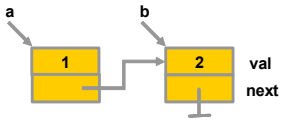
Die drei genannten Datenstrukturen sind nicht zuletzt deshalb für die Software-Entwicklung so bedeutsam, weil hiermit viele Gegenstände der realen Welt gut modelliert werden können. So treten Listen z.B. in einer Lagerhaltung in Form von Einkaufslisten auf, mittels Bäumen lassen sich Hierarchien, wie z.B. die Struktur eines Unternehmens abbilden und Graphen sind zur Darstellung beliebiger Arten von Netzwerken, z.B. einem Verkehrs- oder Kommunikationsnetz, geeignet.

Die Grundlage zur Erstellung und zur Bearbeitung von dynamischen Datenstrukturen bilden die Zeiger, mit denen auf beliebige andere Objekte referenziert werden kann. Dabei ist auch zulässig, dass ein Objekt einer Klasse auf ein Objekt derselben Klasse verweist, was bei allen oben eingeführten Datenstrukturen der Fall ist, da diese nur Objekte einer Klasse beinhalten.

- Der Aufbau und die Manipulation von dynamischen Datenstrukturen erfolgt durch das Verketten von Knoten
  - Verbindung zu einem entfernten Knoten wird durch einen Zeiger (Pointer) auf diesen Knoten realisiert

```
class Node {
  int val;              // value stored in the node
  Node next;            // pointer to another node
  Node(int val) {       // constructor
    this.val = val; next = null;
  }
}
...
Node a = new Node(1); Node b = new Node(2);
a.next = b; b.next = null;
```





Information 21: Verketten von Knoten

Ein Knoten besteht im einfachsten Fall aus einem Wert (hier eine ganze Zahl `int val`, siehe Information 21) und einem Verweis auf einen anderen Knoten (hier `Node next`). Durch Deklaration, Erzeugung und Zeigermanipulationen (z.B. `a.next = b`) lässt sich eine einfache dynamische Struktur aufbauen.

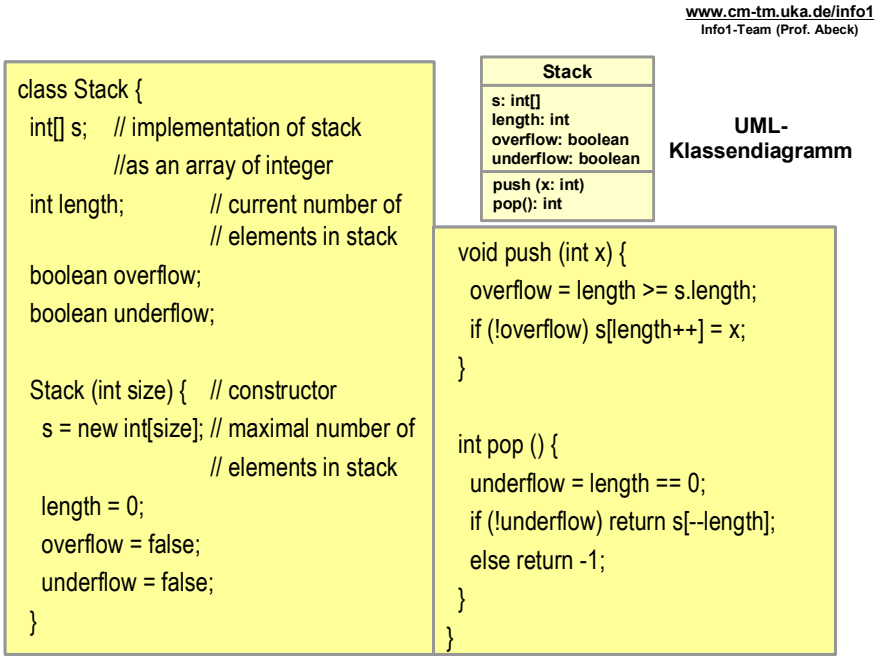
Die Abbildung in Information 21 zeigt die erstellte dynamische Datenstruktur.

3.1 Listen

Die eingeführte Klasse `Node` kann dazu verwendet werden, die in einer Liste zu pflegenden Daten zu halten. Die Liste selbst und die mit dieser Datenstruktur verbundenen typischen Operationen wie

- Einfügen eines neuen Elements am Anfang (*insert*),
- Anhängen eines neuen Elements am Ende (*append*),
- Löschen eines bestehenden Elements (*delete*) oder
- Suchen eines Elements (*search*)

stellen eine eigenständige Abstraktion dar, die in einer separaten Klasse `List` zusammengefasst werden sollte.



Information 22: Listen

Information 22 zeigt das Ergebnis der Modellierung einer Liste in einem UML-Klassendiagramm. Als Attribute werden in List zwei auf Objekte der Klasse Node verweisende Zeiger head und tail vorgesehen, die auf das erste Element bzw. das letzte Element der Liste zeigen sollen.

Den beiden Zeigern werden bei der Erzeugung einer neuen Liste durch entsprechende Initialisierung bei der Deklaration der beiden Attribute die null-Pointer zugewiesen.

Die Methode void insert(int val), die einen Knoten mit dem int-Wert val am Anfang der Liste einfügt, zeigt exemplarisch, wie mit den Zeigern head und tail zu verfahren ist, um das gewünschte Ergebnis zu erhalten.

Das Anhängen eines neuen Elements am Ende (void append(int val)) erfolgt in ganz ähnlicher Form und wird daher an dieser Stelle nicht näher ausgeführt.

3.2 Suchen in einer unsortierten Liste

Bei den Listen gehört das Suchen wie bei den Arrays zu den wichtigsten Operationen. Da beim Einfügen mittels der Methode insert() nicht auf eine bestimmte Reihenfolge geachtet wurde – die Liste also unsortiert ist – muss die Liste sequentiell durchlaufen werden, bis das gesuchte Element gefunden wurde. Falls das Ende der Liste erreicht wird, war die Suche erfolglos.

- Eingaben
  - Liste mit ganzzahligen Werten
  - ganzzahliger Such-Wert
- Ausgaben
  - Zeiger auf das entsprechende Listenelement, falls der Such-Wert in der Liste gefunden wurde
  - null, falls kein Listenelement mit dem Such-Wert in der Liste vorkommt

```
Node search (int val) {
  Node p = head;                    // set p to first element
  _____________________             // traverse list by using p
   _____________________
  // end loop assertion: p == null || p.val == val
  return p;
}
```

Interaktion 23: Suchen eines Listenelements

Dieses Vorgehen der sequentiellen Suche ist im Programm in Interaktion 23 durch eine geeignete Schleifenanweisung zu ergänzen.

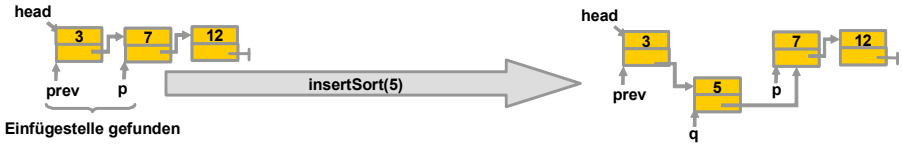
Das Suchen wird beispielsweise in der Methode void delete(int val) zum Löschen eines Elements benötigt, das den angegebenen Wert besitzt.

3.3 Einfügen in eine sortierte Liste

Das Suchen könnte effizienter durchgeführt werden, falls die Liste sortiert wäre. Diese Anforderung wird durch die bisherigen Operationen (insert() bzw. append()) zum Aufbau einer Liste nicht erfüllt, da das Element unabhängig von dessen Wert eingefügt wird.

- Eine sortierte Liste liefert den Vorteil des schnelleren Suchens
  - neue Anforderung: beim Einfügen eines Elements muss die Sortiert-Eigenschaft bestehen bleiben

- Algorithmus-Überlegung anhand eines Beispiels




```
void insertSort (int val) {
  Node p = head; Node prev = null;      // inintialize both traversal pointers
  while (p != null && val > p.val) {    // search insert position
    prev = p; p = p.next; }
  Node q = new Node(val);               // generate node to be inserted
  q.next = p;                           // insert node at the correct position
  if (p == head) head = q; else prev.next = q;
}
```

Information 24: Einfügen in eine sortierte Liste

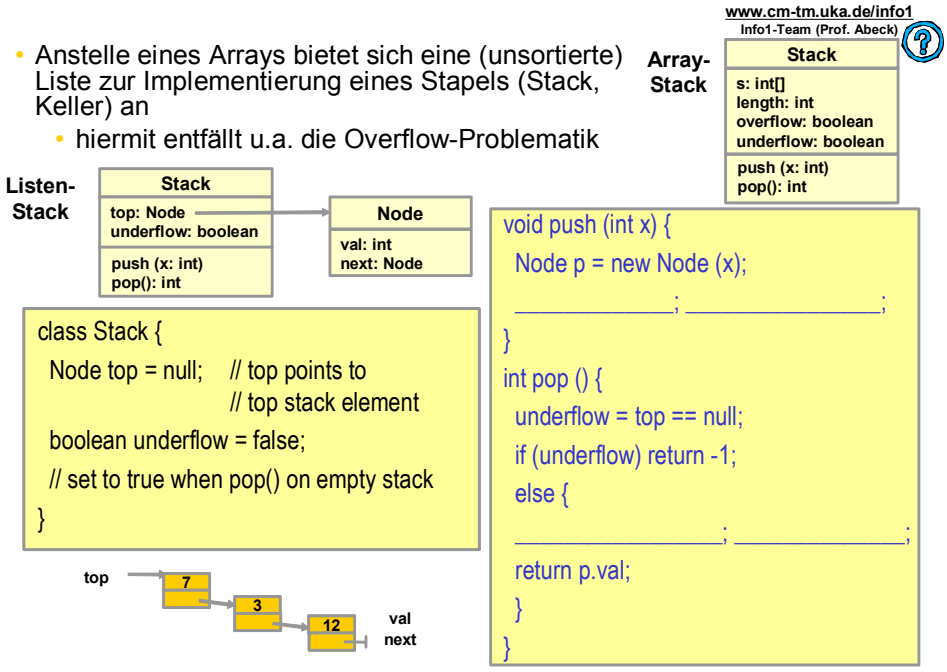
Die Sortiert-Eigenschaft wird nur dann nach dem Einfügen eines neuen Elements erhalten bleiben, wenn zunächst die durch die Sortierung vorgegebene Einfügeposition gesucht wird. An einem einfachen Beispiel kann man sich den Algorithmus zu der Methode insertSort() erarbeiten (siehe Information 24). Offensichtlich ist zum Einhängen des neuen Elements in die Liste nicht nur der Zeiger auf das aktuell überprüfte Element erforderlich (hier p), sondern es wird zusätzlich noch ein weiterer Zeiger benötigt, der das unmittelbar vor dem aktuellen Element liegende Element (prev) in der Liste referenziert. Das Problem in diesem Zusammenhang ist, dass zu einem gegebenen Element immer nur das Nachfolger-Element ermittelt werden kann. In diesem Fall wird aber das Vorgänger-Element benötigt, was nur effizient durch den zusätzlichen Hilfszeiger ermittelt werden kann.

Listen, die nur einen Zeiger auf den Nachfolger vorsehen, heißen einfach verkettet. Werden zu jedem Listenelement sowohl der Nachfolger als auch der Vorgänger in der Datenstruktur gehalten, so heißt die Liste doppelt verkettet.

Die Suche in einer sortierten Liste lässt sich offensichtlich effizienter gestalten, da das sequentielle Durchlaufen im Fall, dass das Element nicht in der Liste enthalten ist, früher abbrechen kann, wie am Beispiel der Arrays in der Kurseinheit IMPERATIVE PROGRAMMIERUNG näher ausgeführt ist.

3.4 Stapel als verkettete Liste

Die Datenstruktur der verketteten Liste bietet eine interessante Alternative zur Realisierung der kennen gelernten Datenstruktur des Stapels (*Stack*, Keller).



Interaktion 25: Stapel als verkettete Liste

Interaktion 25 stellt die *Stack*-Varianten "Array-*Stack*" und "Listen-*Stack*" in Form der UML-Klassendiagramme gegenüber. Beide Datenstrukturen stimmen in ihrem Verhalten insoweit überein, dass sie das gleiche Zugriffsprinzip "*Last In First Out*" (LIFO) realisieren. Der wichtigste Unterschied bzgl. des Verhaltens betrifft das *Overflow*-Problem.

Die push()-Operation wird beim Listen-*Stack* (ganz ähnlich wie die insert()-Operation) in Form einer unsortierten Liste realisiert, da das Stapelelement am Anfang der den Stapel implementierenden Liste eingefügt werden muss. Die Methode pop() liefert das erste Listenelement für den Fall, dass die Liste nicht leer ist, d.h. top != null. Das vom Stapel genommene Element ist nach Ausführung der Methode dann nicht mehr zugreifbar.

In Interaktion 25 sind die in den beiden Zugriffsmethoden durchzuführenden Zeiger-Zuweisungen entsprechend zu ergänzen.

- Die beiden Datenstrukturen Array-Stack und Listen-Stack haben die gleiche Klassen-Schnittstelle
  - (Klassen-) Schnittstelle = Namen der Methoden und deren Parameter
- Eine Klasse mit einer bestimmten Schnittstelle kann auf verschiedene Arten implementiert werden
- Ziel: Austausch der Implementierung einer Klasse (z.B. aus Effizienzgründen) ohne Änderung der Schnittstelle
  - Programme, die diese Klasse nutzen, müssen nicht verändert werden
  - Verstecken der internen Datenstrukturen einer Klasse gegenüber der Außenwelt
    - Information Hiding

Information 26: Schnittstelle und *Information Hiding*

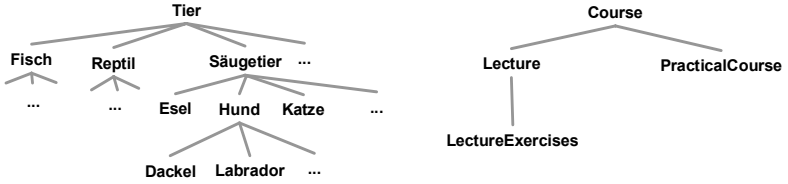
Am Beispiel der beiden Datenstrukturen wird deutlich, dass eine Klassen-Schnittstelle, die durch die Namen der Methoden und deren Parameter bestimmt ist, durch unterschiedliche Implementierungen realisiert werden kann. Die Implementierungen unterscheiden sich dabei üblicherweise bezüglich ihrer Effizienz oder gewisser anderer Merkmale, wie anhand des *Overflow*-Aspekts im obigen Beispiel aufgezeigt wurde.

Mit den Schnittstellen eng verknüpft ist das *Information Hiding*, das ein zentrales Ziel der Objektorientierung darstellt, wie in Information 26 ausgeführt wird

4 VERERBUNG

Zur Objektorientierung gehört das im Folgenden näher beschriebenen Prinzip der Vererbung. Vererbung ist eng mit dem Klassenkonzept verknüpft und stellt eine spezielle Art einer Klassenbeziehung dar.

- Die Vererbung ist eine Art von Beziehung zwischen Klassen
  - eine andere Art von Beziehung ist z.B. das Enthaltensein (Containment)
- Vererbungsbeziehungen sind in der objektorientierten Programmierung mit Umsicht zu nutzen
  - können das Programm komplex und damit schwer durchschaubar machen
- Die Vererbungsbeziehung tritt an vielen Stellen in der Wirklichkeit auf
  - Klassen dienen dazu, Dinge der realen Welt zu modellieren
- Beispiele für Vererbungshierarchien



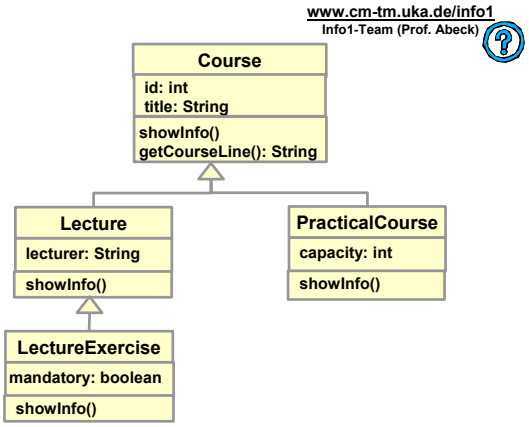
Information 27: VERERBUNG - Einführung und Beispiele

Vererbungsbeziehungen treten an vielen Stellen in der Realität auf, wie an den zwei unterschiedlichen Beispielen von Vererbungshierarchien in Information 27 verdeutlicht wird.

4.1 Modellieren und Programmieren von Vererbungsbeziehungen

Die *Unified Modeling Language* (UML) ermöglicht die Modellierung von Vererbung durch die Kennzeichnung einer Klassenbeziehung mittels einer hohlen Pfeilspitze. Dabei ist die Klasse, auf die die Pfeilspitze zeigt, die so genannte Oberklasse. Die andere mit der Oberklasse in der Vererbungsbeziehung stehende Klasse heißt Unterklasse.

- Modellierung einer Vererbungsbeziehung in einem UML-Klassen-diagramm durch eine hohle Pfeilspitze
  - Attribute und Methoden der Oberklasse, auf die die Pfeilspitze zeigt, werden der Unterklasse vererbt



- Programmierung einer Vererbungsbeziehung in Java durch das Schlüsselwort extends

```
class Course { ... }
class Lecture extends Course { ...}
class LectureExercise extends _____ { ...}
class PracticalCourse extends _____ { ... }
```

Interaktion 28: Modellieren und Programmieren von Vererbungsbeziehungen

Im Beispiel in Interaktion 28 übernimmt die Unterklasse Lecture die Attribute id und title sowie die Methoden showInfo() und getCourseLine() von deren Oberklasse Course. Während die Methode showInfo() die Attributwerte auf dem Bildschirm ausgibt, erzeugt getCourseLine() einen String, der als Eintrag für ein Veranstaltungsverzeichnis dient.

Gegenüber der Klasse LectureExercise ist Lecture ihrerseits eine Oberklasse. Das bedeutet, dass LectureExercise alle Attribute und Methoden von Lecture erbt – das sind in diesem Fall sowohl die in Lecture definierten Attribute (lecturer) und Methoden (showInfo()) als auch die von Course geerbten Attribute und Methoden. Außerdem wird in LectureExercise ein boolesches Attribut mandatory definiert, das true ist, falls die zu der Vorlesung angebotene Übung verpflichtend (*mandatory*) ist (weil beispielsweise von den Teilnehmern ein Pflichtschein erworben werden muss).

Die Vererbungsbeziehung zwischen zwei Klassen lässt sich in Java einfach durch Verwendung des Schlüsselworts extends ausdrücken, wie der in Interaktion 28 zu ergänzende Programmausschnitt verdeutlicht.

Einen praktischen Nutzen der Vererbungsbeziehung kann man sich anhand der vererbten Methode getCourseLine() klarmachen: Diese Methode wird nur einmal in der Oberklasse Course definiert und kann aufgrund der Vererbungsbeziehung in allen Unterklassen (Lecture, LectureExercises, PracticalCourse) verwendet werden, um zu jeder dieser verschiedenen Veranstaltungen den Verzeichniseintrag bestehend aus id und title zu erzeugen.

Anders verhält es sich mit der Methode showInfo(), die in den Unterklassen überschrieben wird. Es lässt sich leicht nachvollziehen, dass die Methode showInfo() zu einem Objekt der Klasse Lecture nicht nur die Information zu seiner Oberklasse Course, sondern auch den Namen des Dozenten ausgeben möchte, der im Attribut lecturer gespeichert ist.

- Von einer Oberklasse geerbte Methoden können in der Unterklasse überschrieben werden
  - Überschreiben heißt, dass die Methode mit der gleichen Schnittstelle neu deklariert wird
  - Beispiel: Methode showInfo()

- In der überschreibenden Methode einer Unterklasse kann mittels eines speziellen Aufrufs super.<Methodennnamen> auf die (überschriebene) Methode der Oberklasse zugegriffen werden

```
class Lecture extends Course {
  ...
  void showInfo() {              // Lecture method which is overwriting Course method
    super.showInfo();            // calls showInfo() from class Course
    Out.println("Lecturer: " + lecturer);
    ....
  }
}
```

Information 29: Methoden und Vererbungsbeziehung

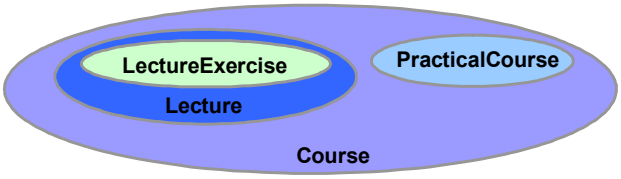
Wie der Programmausschnitt in Information 29 zeigt, kann in der überschreibenden Methode zum Objekt der Unterklasse (*Sub Class*) die überschriebene Methode zu dem Objekt der Oberklasse (*Super Class*) durch einen so genannten super-Aufruf in Anspruch genommen werden. Hierdurch lässt sich in der Beispiel-Methode showInfo() zu einem Lecture-Objekt zunächst die Information zum Course-Objekt, dessen Eigenschaften geerbt wurden, anzeigen (super.showInfo()) und um die zusätzlich anzuzeigende Information (hier lecturer) ergänzen.

4.2 Polymorphie und Kompatibilität

Durch die Vererbung wird bewirkt, dass Attribute und Methoden der Oberklasse automatisch auch zur Unterklasse gehören. Hieraus resultiert, dass eine Objektvariable der Unterklasse so auftreten kann, als wäre sie ein Objekt der Oberklasse. Die Objektvariable kann also in Gestalt verschiedener Klassen auftreten. Diese mit der Objektorientierung verbundene Eigenschaft der Vielgestaltigkeit wird auch als Polymorphie bezeichnet.

- Der Nutzen der Vererbung liegt darin, dass eine Unterklasse mit einer Oberklasse kompatibel ist
  - jedes Programm, das in der Lage ist, mit Objekten der Oberklasse zu arbeiten, kann auch mit Objekten der Unterklasse arbeiten
  - Objekte können polymorph (vielgestaltig) sein

- Die Kompatibiltät gilt, weil jedes Objekt einer Unterklasse zugleich auch ein Objekt der Oberklasse ist
  - "ist ein"-Beziehung (englisch "is a")
  - Objekt der Unterklasse "ist ein" Objekt der Oberklasse
  - in umgekehrter Richtung gilt die "ist ein"-Beziehung nicht



Information 30: Kompatibilität zwischen Oberklasse und Unterklasse

Die Eigenschaft der Polymorphie ist aufgrund der Kompatibilität zwischen Oberklasse und Unterklasse gegeben. Die Abbildung in Information 30 zeigt den Zusammenhang der Objekte der oben eingeführten Klassen in Form eines Mengendiagramms. Das Diagramm besagt, dass z.B. die Menge der LectureExercise-Objekte in der Menge der Lecture-Objekte enthalten ist, die ihrerseits Untermenge der Course-Objekte ist. Diese Mengenbeziehung zwischen Objekten der Oberklasse und Unterklasse wird durch die "ist ein"-Beziehung ausgedrückt.

- Einer Objektvariablen der Oberklasse kann ein Objekt der Unterklasse zugewiesen werden
  - diese Typflexibilität gilt auch für Parameterübergaben
  - Typkonversion zur Unterklasse möglich

```
Course a = new Lecture();
...
if (a instanceof Lecture)
  Lecture b = (Lecture) a;
```

- Zwei Arten von Typen bei Variablen zu unterscheiden

(1) statischer Typ
  - Typ, in der die Variable deklariert wurde
  - im Beispiel: statischer Typ von a ist Course

(2) dynamischer Typ
  - Typ des Objekts, auf das die Variable zur Laufzeit zeigt
  - im Beispiel: dynamischer Typ von a ist nach der Zuweisung Lecture

Information 31: Statischer und dynamischer Typ

Die Kompatibilitäts-Eigenschaft bewirkt, dass einer Objektvariablen, die als Typ der Oberklasse deklariert wurde, ein Objekt der Unterklasse zugewiesen werden kann, wie der Programmausschnitt in Information 31 verdeutlicht.

Hieraus resultiert, dass bzgl. der Frage, von welchem Typ diese Variable ist, zwischen einem statischen Aspekt, der durch die Deklaration gegeben ist und einem dynamischen Aspekt, der durch die Zuweisung ggf. kompatibler Typen zur Laufzeit festgelegt wird, unterschieden werden muss.

4.3 Dynamische Bindung

Im Zusammenhang mit der Polymorphie stellt sich unmittelbar die Frage, die Methode welcher Klasse aufgerufen wird, wenn eine polymorphe Variable eine entsprechende Meldung zum Aufruf einer Methode erhält.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

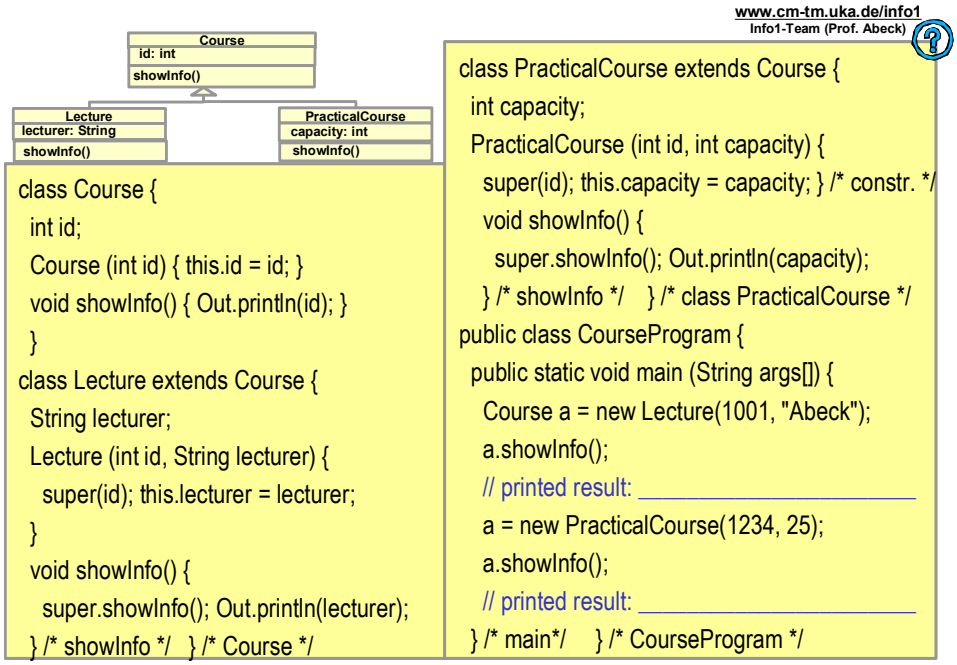
- Eine Meldung obj.m() führt immer zum Aufruf der m()-Methode, die zum dynamischen Typ von obj gehört
  - die Meldung wird also dynamisch (d.h. zur Laufzeit) gebunden

- Vorteile
  - der Aufrufer muss sich nicht darum kümmern, von welchem Typ eine polymorphe Variable gerade ist
  - es können Unterklassen ergänzt werden, ohne den Teil des Programms ändern zu müssen, durch den der Aufrufer realisiert wird

Information 32: Dynamische Bindung

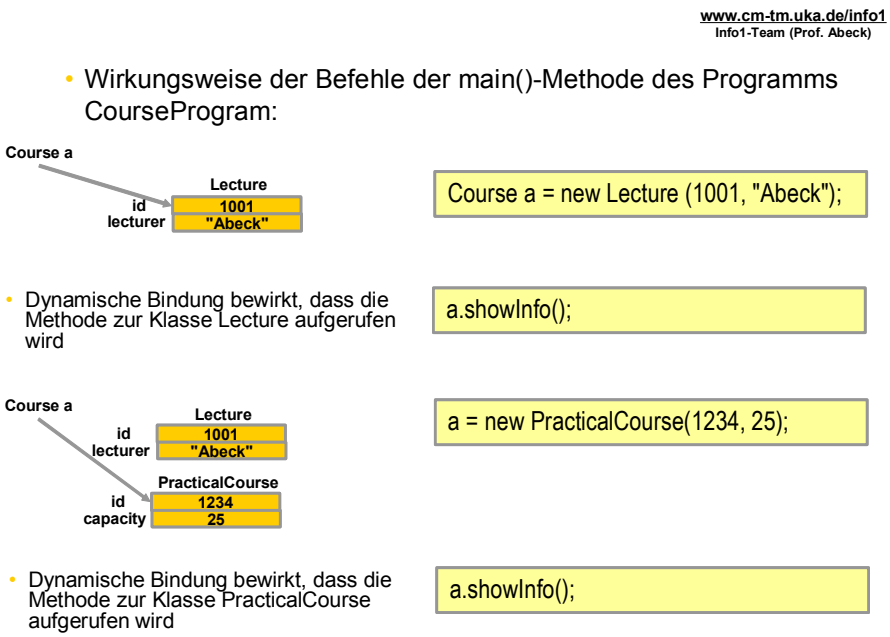
Die Antwort lautet, dass der dynamische Typ der Variablen zugrunde gelegt wird; die Variable wird dynamisch – also zur Laufzeit – an die Klasse gebunden. Im Beispielprogramm von Information 31 heißt das konkret, dass eine Meldung a.showInfo() zum Aufruf der showInfo()-Methode der Klasse Lecture führt.

Die dynamische Bindung hat zwei große Vorteile bei der Erstellung von Programmen, die mit polymorphen Objekten umgehen, die in Information 32 beschrieben sind.



Interaktion 33: Beispiel zur dynamischen Bindung von Methodenaufrufen

Das Programm in Interaktion 33 greift das zuvor eingeführte Beispiel zu den Lehrveranstaltungen (Klassen Course, Lecture, PracticalCourse) in reduzierter Form auf. Neben der dynamischen Bindung der Objektvariablen a in der Methode main() wird in den Klassendefinitionen zu Lecture und PracticalCourse der Zugriff auf die Konstruktor-Methode mittels super(id) und die Methode showInfo() mittels super.showInfo()) verdeutlicht.



Information 34: Veranschaulichung des Ablaufs

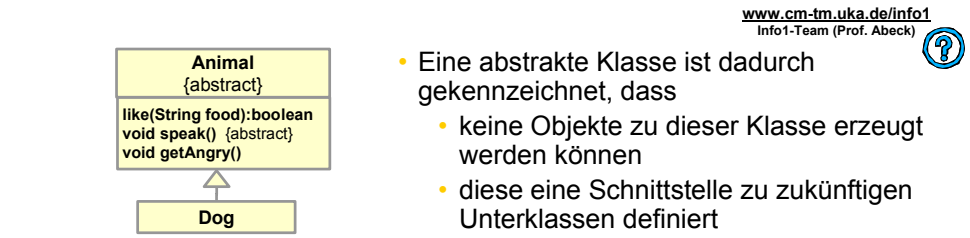
In Information 34 wird die Wirkungsweise der in der main()-Methode enthaltenen Befehle graphisch veranschaulicht. Die Zuweisungen zur Variablen a der Klasse Course zeigen, dass die

Variable auf Objekte von Unterklassen zu Course – hier sind das die Klassen Lecture und PracticalCourse – zeigen kann.

Wird die überladene Methode showInfo() aufgerufen, so bewirkt die dynamische Bindung, dass nicht die Methode der (statisch durch Deklaration bestimmten) Klasse Course, sondern die Methode der (dynamisch zur Laufzeit zugewiesenen) Klassen Lecture bzw. PracticalCourse aufgerufen wird.

4.4 Abstrakte Klassen und Interfaces

In Vererbungshierarchien kann der Fall auftreten, dass man zu einer Superklasse die Schnittstelle, aber nicht deren Implementierung vollständig angeben kann. Zu einer solchen Superklasse soll auch kein Objekt erzeugt werden können, d.h., die Klassenbeschreibung soll ausschließlich als Grundlage für die hieraus abgeleiteten Unterklassen dienen.




```
abstract class Animal {
  boolean like(String food) { return false; }
  abstract void speak();
  void getAngry() { speak(); }
}
class Dog extends Animal {
  boolean like(String food) {
    return food.equals("bones"); }
  void speak() {Out.println("bark");}
  void getAngry() {Out.println("growl");}
}
```

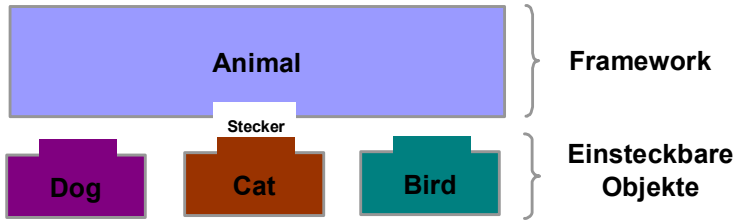
```
public class AnimalProgram {
  public static void main (String args[]) {
    Animal a = new Dog();
    if (a.like("corn")) a.speak(); else a.getAngry();
    // printed result: _____
    if (a.like("bones")) a.speak(); else a.getAngry();
    // printed result: _____
  }
}
```

Interaktion 35: Abstrakte Klassen

Besitzt eine Klasse die beschriebenen Anforderungen, so ist sie als so genannte abstrakte Klasse zu deklarieren. Im UML-Klassendiagramm werden abstrakte Klassen durch {abstract} unter dem Klassennamen gekennzeichnet. Die Sprache Java sieht das Schlüsselwort abstract vor, das vor dem Schlüsselwort class anzugeben ist.

Diejenigen Methoden, zu denen in einer abstrakten Klasse keine Implementierungen erfolgen, sind ebenfalls mit diesem Schlüsselwort zu kennzeichnen. Im Beispiel in Interaktion 35 wird die Methode speak() der abstrakten Klasse Animal als abstrakte Methode definiert. Abstrakte Methoden müssen zwingend in den Unterklassen überschrieben werden. Im Beispiel überschreibt die Unterklasse Dog nicht nur diese Methode, sondern auch die beiden anderen (nicht abstrakten) Methoden like() und getAngry().

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)



- Eine abstrakte Klasse definiert eine Familie von Klassen, die alle eine bestimmte Menge von Meldungen verstehen
  - eine Variable der abstrakten Klasse wirkt wie ein Steckplatz, in die die Objekte dieser Familie eingesteckt werden können
- Eine Software mit solchen Steckplätzen wird als ein Framework bezeichnet
  - sind Halbfabrikate, die durch Einstecken von Objekten zu verschiedenen Endfabrikaten ausgebaut werden können

Information 36: *Framework*-Konzept

Die Abbildung in Information 36 verdeutlicht den Zusammenhang zwischen der abstrakten Klasse (hier Animal) und den daraus abgeleiteten Unterklassen (hier Dog, Cat und Bird). Die von der abstrakten Klasse vorgegebenen Methoden stellen eine Art Stecker (Plug) dar, in den die Objekte der Unterklassen eingesteckt werden können, um die unvollständige abstrakte Klasse zu vollenden.

Eine Software, die gemäß diesem Steckprinzip aufgebaut ist, wird als *Framework* (Rahmenwerk) bezeichnet.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Abstrakte Klassen können neben abstrakten Methoden auch nicht-abstrakte Methoden enthalten
  - sind alle Methoden abstrakt, entspricht die abstrakte Klasse einem Interface (Klassenschnittstelle)
- Ein Interface kann neben abstrakten Methoden Konstanten-Attribute enthalten, aber keine Variablen-Attribute und keine Konstruktoren

```
interface Writer {
  int eol = '\n';
  void open();
  void close();
  void write(char ch);
}
```

- Alle Methoden eines Interface sind automatisch öffentlich (public) und abstrakt (abstract)
- Alle Attribute sind automatisch öffentlich (public) und konstant (static final)

- Ein Interface kann zu einem "Unter-Interface" erweitert werden

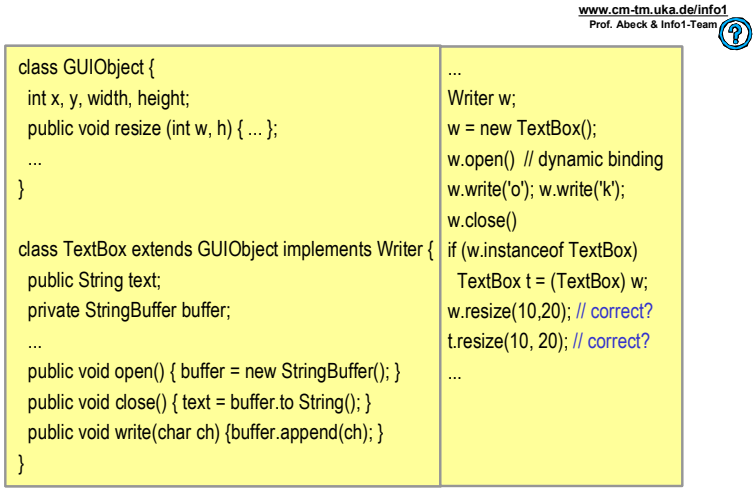
```
interface ReadWriter extends Writer { char read(); }
```

Information 37: *Interfaces*

Eine spezielle Ausprägung von abstrakten Klassen stellen die *Interfaces* (Schnittsllen) dar. Die besonderen Eigenschaften von *Interfaces* sowie ein Beispiel interface Writer sind in Information 37 beschrieben.

Eine Klasse kann von einem *Interface* (Schnittstellen-Klasse) erben und muss in diesem Fall Implementierungen zu allen Methoden des Interfaces liefern, da *Interface*-Methoden gemäß Definition abstrakt sind.

Interfaces sind spezielle Klassen und bieten daher auch die Möglichkeit der Erweiterung, also der Bildung von "Unter-*Interfaces*".



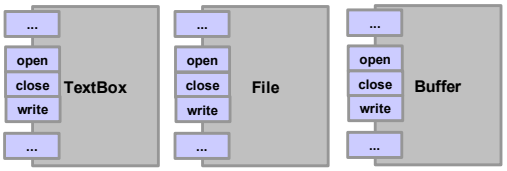
Interaktion 38: Beispiel zur Nutzung des *Interface* Writer

Im Beispiel in Interaktion 38 erbt (extends) die Klasse TextBox von der Klasse GUIObject und implementiert (implements) das Interface Writer. Alle *Interface*-Methoden von Writer müssen in TextBox implementiert werden. Die Implementierung muss dabei nicht zwingend direkt in der Klasse erfolgen (wie das im Beispiel gegeben ist), sondern kann auch durch Erben von einer anderen Klasse erfolgen.

Da *Interfaces* Typen sind, kann man auch entsprechende Variablen deklarieren (z.B. Writer w). Da ein *Interface* eine spezielle Form einer abstrakten Klasse darstellt, verhält sich eine Klasse, die ein *Interface* implementiert, wie dieses *Interface* (Polymorphie). Im Beispiel ist es daher gestattet, ein TextBox-Objekt in einer Writer-Variablen zu speichern (w = new TextBox()). Eine Meldung an dieses TextBox-Objekt (z.B. w.open() oder w.write()) wird entsprechend dynamisch gebunden. Der Zugriff auf Methoden, die in der TextBox-Klasse definiert wurden oder die von anderen Klassen geerbt wurden, ist nur nach geeigneter Typ-Konversion möglich. Dieser Aspekt ist bei der Beantwortung der in Interaktion 38 gestellten Fragen zu berücksichtigen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Interfaces ermöglichen die Gleichbehandlung von Klassen, die in keiner Vererbungsbeziehung stehen




```
Writer w = new File(...);
w.open(); // calls open() from File
...
w = new Buffer(...);
w.open(); // calls open() from Buffer
```

- Eine Klasse kann beliebig viele Interfaces implementieren

```
class Buffer extends StorageObject implements Writer, Comparable, Serializable { ... }
```

- Im Gegensatz zur Mehrfach-Implementierung von Interfaces ist die Mehrfach-Vererbung kritisch und wird daher in Java nicht unterstützt
  - Grund: Zwei Basisklassen könnten Methoden mit gleichem Namen und gleicher Schnittstelle aufweisen
  - Kein Problem bei Interfaces, weil die Methode in der Unterklasse neu implementiert werden muss

Information 39: Einsatzzweck von Interfaces

Interfaces vererben somit nur den Typ und nicht die eigentliche Implementierung. Hierdurch lassen sich Klassen im Sinne der Polymorphie gleich behandeln, die in keiner direkten Beziehung – insbesondere in keiner Vererbungsbeziehung – stehen. Information 39 verdeutlicht diesen Sachverhalt an einem Beispiel mit drei Klassen TextBox, File und Buffer.

Interfaces bieten außerdem eine Möglichkeit, die Eigenschaft auszudrücken, dass eine Klasse Eigenschaften von mehreren Klassen erbt. Diese als Mehrfach-Vererbung (*Multiple Inheritance*) bezeichnete Eigenschaft ist in Java aufgrund des in Information 39 genannten Problems nicht erlaubt.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Aufbau und Einsatz von Frameworks
  - die sinnvolle Verwendung von Vererbung ist bis heute noch nicht abschließend geklärt
- Komponentenbasierte Programmierung
  - gezielte Nutzung von Interfaces (Beispiel: JavaBeans)
- Entwurfsmuster (Patterns)
  - schematische Lösung häufig auftretender Probleme in Softwarearchitekturen mittels objektorientierter Techniken
- Weitere Java-Sprachmittel (z.B. geschachtelte Klassen)
- Klassenbibliotheken (z.B. zur Unterstützung von Applets, XML, ...)

Information 40: Ausblick auf weitere Themen der Objektorientierung

Die behandelten Konzepte zur Vererbung liefern eine gute Grundlage, dieses mächtige aber auch nicht ganz unproblematische objektorientierte Konzept in der Programmierung vorteilhaft zu nutzen. Information 40 macht deutlich, dass die Beschreibung dieses Kapitels über die Vererbung bei weitem nicht vollständig ist.

Es sei abschließend angemerkt, dass das Konzept der Objektorientierung nicht nur in der Programmierung, sondern verstärkt auch in der Modellierung genutzt wird, wie in der Kurseinheit GRUNDBEGRIFFE DER INFORMATIK ausgeführt wurde. In zahlreichen weiterführenden Veranstaltungen, die sich mit dem für die Informatik zentralen Themen der Modellierung und der Softwareentwicklung beschäftigen, wird die Objektorientierung aufgegriffen und weiter vertieft.

VERZEICHNISSE

Abkürzungen und Glossar

Abkürzung oder Begriff	Langbezeichnung und/oder Begriffserklärung
Dynamische Datenstruktur	Datenstruktur die sich dynamisch zur Laufzeit aufbauen und manipulieren lässt. Beispiele dynamischer Datenstrukturen sind Listen, Bäume oder Graphen.
einfach verkettet	Typ einer Liste, die einen Zeiger auf den Nachfolger vorsieht (aber keinen Zeiger auf den Vorgänger).
Framework	Eine Software, die so aufgebaut ist, dass Software-Baustein gemäß einem auf dem Vererbungsprinzip basierenden Steckprinzip ergänzt werden können.
Interface	Eine spezielle Ausprägung einer abstrakten Klasse, die abstrakte Methoden und Konstanten-Attribute beinhaltet.
Information Hiding	Zentrales Prinzip der Objektorientierung, durch das die internen Datenstrukturen einer Klasse gegenüber der Außenwelt versteckt werden.
ISWA	INTERNET-SYSTEME UND WEB-APPLIKATIONEN Titel einer Lehrveranstaltung.
K&D	KOMMUNIKATION UND DATENHALTUNG Titel einer Lehrveranstaltung.
Klasse	Einheit bestehend aus Daten (Attribute) und darauf arbeitende Operationen (Methoden). Klassen stellen eine geeignete Form zur Gliederung komplexer Softwaresysteme dar.
Konstruktor	Spezielle Klassenmethode, die bei der Objekterzeugung dazu genutzt werden kann, ein Objekt in der gewünschten Form zu initialisieren.
LIFO	*Last In First Out* Ein auch als Kellerprinzip bezeichnetes Zugriffsprinzip, das besagt, dass das zuletzt geschriebene Element (*Last In*) bei der nächsten Leseoperation ausgegeben wird (*First Out*).
null	Nullpointer Wert, der besagt, dass auf keinen Inhalt gezeigt wird. Beispiel: Eine bereits deklarierte Klassenvariable, der aber noch kein Objekt zugewiesen wurde, hat den Wert null.
Objektattribut	Einem speziellen Objekt zugeordnetes Attribut, das erst zur Zeit der Objekterzeugung angelegt wird.
Objektorientierte Programmierung	Philosophie des Programmierens, die von einer Welt ausgeht, die aus gleichberechtigten und einheitlich erscheinenden Objekten besteht. Oberstes Prinzip des objektorientierten Vorgehens ist es, Objekte stets nur von außen

zu betrachten und ihren inneren Aufbau zu ignorieren [CS93].

Polymorphie — Eine im Zusammenhang mit der Objektorientierung auftretende Vielgestaltigkeit in der Form, dass eine Objektvariable der Unterklasse so auftreten kann, als wäre sie ein Objekt der Oberklasse. Die Objektvariable kann also in Gestalt verschiedener Klassen auftreten.

Stapel — Bezeichnung einer Rechenstruktur, der das LIFO-Prinzip (*Last In First Out*) zugrunde liegt.
Synonymer Begriff: Keller
Englischer Begriff: *Stack*

UML — *Unified Modeling Language*
Sprache, die aus graphischen Elementen besteht und zur semi-formalen Beschreibung von beliebigen Gegenständen (z.B. Software-Systeme oder Geschäftsbereiche) genutzt wird.

Vererbung — Objektorientiertes Prinzip, das eng mit dem Klassenbegriff verknüpft ist und eine von verschiedenen Arten von Beziehungen zwischen Klassen darstellt, durch die Attribute und Operationen einer Klasse an eine Unterklasse weitergegeben („vererbt“) werden.

Virtuelle Maschine — Ein in Software realisierter Prozessor.

Index

Informationen und Interaktionen

Literatur

[C&M-ISWA] Cooperation&Management, Kursdokumente zur Vorlesung "INTERNET-SYSTEME UND WEB-APPLIKATIONEN (ISWA)", http://www.cm-tm.uka.de/iswa, Universität Karlsruhe (TH), C&M (Prof. Abeck)

[C&M-KuD] Cooperation&Management, Kursdokumente zur Vorlesung "KOMMUNIKATION UND DATENHALTUNG (K&D)", http://www.cm-tm.uka.de/iswa, Universität Karlsruhe (TH), C&M (Prof. Abeck)

[CS93] Duden "Informatik" – ein Sachlexikon für Studium und Praxis, Dudenverlag, 1993.

[Ma89] Udi Manber: Introduction to Algorithms – A Creative Approach, Band 1: Programmierung und Rechstrukturen, Addison Wesley, 1989.

[Me90] Bertrand Meyer: Objektorientierte Softwareentwicklung, Carl Hanser Verlag, 1990.

[Mö03] Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

[Oe01] Bernd Oestereich: Objektorientierte Software-Entwicklung – Analyse und Design mit der Unified Modeling Language, Oldenbourg Verlag, 2001.

FORTGESCHRITTENE PROGRAMMIERKONZEPTE

Kurzbeschreibung

In dieser Kurseinheit wird mit der Ausnahmebehandlung und den *Threads* auf zwei wichtige fortgeschrittene Konzepte der Programmierung eingegangen und deren Umsetzung in Java wird aufgezeigt.

Schlüsselwörter

Ausnahme, Fehlercode, Ausnahmebehandlung, Klasse Exception, try-catch-Konstrukt, Parallelität, *Thread*, Klasse Thread, Synchronisation, kritischer Bereich, Sperrvariable

Lernziele

1. Die Vorteile eines Ausnahmebehandlungskonzepts gegenüber Fehlercodes werden verstanden und die hierfür bereitgestellten Sprachelemente können bei der Erstellung von Programmen gezielt eingesetzt werden.
2. Das Konzept der *Threads* sowie deren Synchronisation werden durchdrungen und die Umsetzung dieses Konzepts in einfache Programme wird beherrscht.

Hauptquellen

- Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

Inhaltsverzeichnis

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- AUSNAHMEBEHANDLUNG
  - Grundlegende Konzepte, Arten von Ausnahmen, Klasse Exception, Auslösen und Behandeln einer Ausnahme
- THREADS
  - Parallelität, Klasse Thread, Erzeugung, Synchronisation, synchronized-Anweisung, kritischer Bereich, Deadlock, Monitor

Information 1: FORTGESCHRITTENE PROGRAMMIERKONZEPTE

1 AUSNAHMEBEHANDLUNG

Ein nicht zu unterschätzender Teil eines Programms besteht darin, Fehlersituationen zu behandeln, die z.B. aufgrund eines kurzfristig entstandenen Ressourcenmangels oder einer fehlerhaften Benutzereingabe im Programmverlauf aufgetreten sind [Mö03].

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Im Ablauf eines Programms können vielfältige Fehlersituationen auftreten, die vom Programm zu behandeln sind
- Zwei Möglichkeiten des Umgangs mit Fehlersituationen
  1. Einführung von Fehlercodes, die von Methoden im Falle einer festgestellten Fehlersituation zurückgeliefert werden
  2. Konzepte zur Ausnahmebehandlung (Exception Handling), die von modernen Programmiersprachen angeboten werden

Information 2: AUSNAHMEBEHANDLUNG - Problemstellung und Lösungsansätze

Grundsätzlich bestehen die zwei in Information 2 genannten Möglichkeiten zum Umgang mit den verschiedenen Fehlerarten.

Bevor das Konzept der Ausnahmebehandlung am Beispiel der Sprache Java vorgestellt wird, soll zunächst eine konventionelle Technik – die Einführung von Fehlercodes – behandelt werden. Aus den Nachteilen der Fehlercodes wird die dann ausführlicher beschriebene Ausnahmebehandlung motiviert.

1.1 Fehlercodes

Die Idee der Fehlercodes besteht darin, dass Methoden eventuell festgestellte Fehler an die aufrufende Methode in Form eines Rückgabewertes melden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Fehlercodes sind Funktionswerte, die von einer Methode zurück geliefert werden
- Beispiel eines Fehlercodes
  - Methode enterCourse() liefert als Ergebnis
    - 0 (ok)
    - 1 (overflow)
    - 2 (numberExists)
    - 3 (...)
- Problem: Fehlerbehandlung verschlechtert die Lesbarkeit des Programms bei mehreren Methodenaufrufen erheblich

```
result = f();
if (result == ok) {
  result = g();
    if (result == ok) {
      result = h();
      if (result == ok) {
        ...
      } else ... // error handling
    } else ... // error handling
  } else ... // error handling
}
```

Wie würde das Programm ohne Abfrage von Fehlercodes lauten?

Interaktion 3: Einführung von Fehlercodes

Es kann beispielsweise für die Methode enterCourse() folgender Fehlercode vereinbart werden:

- ok: es ist kein Fehler in der Methode aufgetreten.
- overflow: der Kurs konnte nicht in den Katalog aufgenommen werden, weil die maximale Kapazitätsauslastung des Kurskatalogs erreicht ist.
- numberExists: die Kursnummer ist bereits vorhanden, weshalb eine Aufnahme des Kurses in den Katalog nicht möglich ist.

Die Fehlercode-Technik weist den großen Nachteil auf, dass der eigentliche (fehlerfreie) Ablauf des Programms durch die nach jedem Methodenaufruf erforderliche Abprüfung des Fehlercodes nicht mehr klar erkennbar ist. Das Beispielprogramm in Interaktion 3, das die Abfolge von drei Methodenaufrufen skizziert, macht dieses Problem deutlich.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

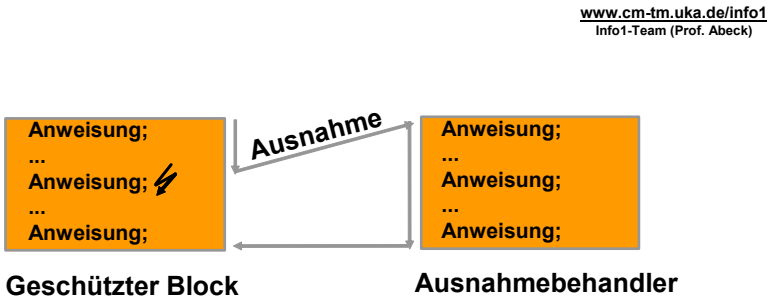
- Trennung der Fehlerbehandlung vom Normalablauf
  - Fehlermeldung nicht über Rückgabewerte zu realisieren, die eine rufende Methode zu einer Fehlercode-Überprüfung zwingt
- Flexibilität der Fehlerbehandlung
  - eine beliebige Methode in der Aufrufkette soll auf eine festgestellte Fehlersituation reagieren können
- Garantierte Fehlerbehandlung
  - jeder mögliche Fehler wird garantiert von irgendeiner Methode in der Aufrufkette bearbeitet

Information 4: Anforderungen an die Fehlerbehandlung

Aus den Nachteilen, die durch die Fehlercodes entstehen, resultiert die erste, in Information 4 genannte Anforderung der Trennung der Fehlerbehandlung vom Normalablauf, der die eigentliche Algorithmusbeschreibung darstellt. Die zwei weiteren Anforderungen der Flexibilität und der garantierten Bearbeitung von festgestellten Fehlersituationen führen zum Konzept der Ausnahmebehandlung (*Exception Handling*).

1.2 Trennung der Fehlerbehandlung

Die notwendige Trennung der Fehlerbehandlung erfolgt durch die Einführung von so genannten Ausnahmebehandlern (*Exception Handler*).



- Ein Block kann vor dem Auftreten von Fehlern geschützt werden
  - tritt ein Fehler auf, wird eine Ausnahme (exception) erzeugt
  - die Ausnahme wird durch einen Ausnahmebehandler bearbeitet
  - danach wird die Bearbeitung des Programms an der Stelle nach dem geschützten Block fortgesetzt

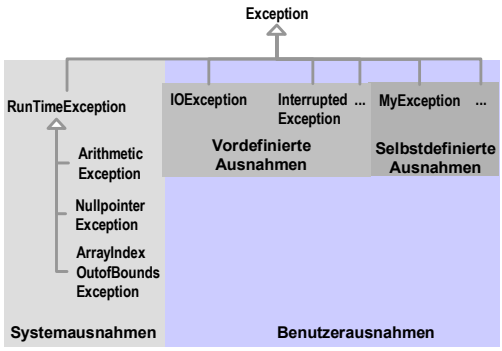
Information 5: Konzept der Ausnahmebehandlung

Ein Block, in dem eine Fehlersituation auftreten kann, wird durch ein geeignetes Sprachelement geschützt. Tritt eine Ausnahme tatsächlich auf, wird der entsprechende für diese Ausnahme zuständige Ausnahmebehandler aufgerufen. Nach Behandlung der Ausnahme setzt das Programm mit der ersten Anweisung hinter dem geschützten Block (*Protected Block*) fort. Den Programmfluss bei Auftreten einer Ausnahme im geschützten Block zeigt die Abbildung in Information 5.

- Der Programmausschnitt zeigt die einfachste Form einer try-Anweisung

- Arten von Ausnahmen in Java
  - Systemausnahmen
    - ArithmeticException
    - NullPointerException
    - ArrayIndexOutofBounds Exception
  - Benutzerausnahmen
    - durch eine throw-Anweisung ausgelöst, um Fehler zu signalisieren
    - vordeklarierte (z.B. java.io.IOException) und vom Programmierer selbst definierte Ausnahmen (z.B. MyException)

```
try {
  ...          // protected block
} catch (Exception e) {
  ...          // exception handler
}
```



Information 6: try-Anweisung und Arten von Ausnahmen

In Java steht als Sprachelement zur Ausnahmebehandlung die try-Anweisung zur Verfügung, deren einfachste Ausprägung im Programmausschnitt in Information 6 gezeigt wird. Hierdurch lässt sich eine Ausnahme im geschützten Block "einfangen" (catch()) und die Bearbeitung dieser Ausnahme kann in einem nachfolgenden Ausnahmebehandler-Block durchgeführt werden.

Es werden in Java zwei Arten von Ausnahmen – die Systemausnahmen und die Benutzerausnahmen – unterschieden:

- Systemausnahmen entstehen aufgrund von illegalen, vom Programm ausgeführten Instruktionen, wie z.B. die Division durch 0 (ArithmeticException), der Zugriff auf ein Objekt über einen Null-*Pointer* (NullPointerException) oder ein außerhalb der Indexgrenzen liegender Array-Zugriff (ArrayIndexOutOfBoundsException).

- Benutzerausnahmen werden durch unzulässige Eingaben des Benutzers verursacht. Sie werden explizit durch die weiter unten behandelte throw-Anweisung ausgelöst. Neben den von Java vordefinierten Benutzerausnahmen, wie z.B. eine bei der Ein-/Ausgabe auftretende Ausnahme (java.io.IOException) kann der Programmierer eigene, d.h. selbst definierte Benutzerausnahmen einführen.

1.3 Klasse Exception

Hinter Ausnahmen verbirgt sich eine Klasse Exception, die als Datenanteil Informationen über den aufgetretenen Fehler beinhaltet und als Funktionsanteil verschiedene Methoden zum Zugriff auf diese Informationen bereitstellt. Die oben beschriebenen Ausprägungen von Ausnahmen entstehen dadurch, dass von der Klasse Exception Unterklassen abgeleitet werden. Die hieraus resultierende Klassenhierarchie zeigt die Abbildung in Information 6.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
class Exception {
  String toString();              // describes type of exception
  void printStackTrace();         // prints chain of called methods
  ...
}
```

- Basisklasse (Wurzel) der Vererbungshierarchie
  - Daten und Methoden werden von allen Exception-Unterklassen übernommen und ergänzt
- Zugriff über ein Exception-Objekt e
  - e.toString() beschreibt die Art des Fehlers und kann in Fehlermeldungen verwendet werden
  - e.printStackTrace() gibt die Methodenaufrufkette bis zur Methode des Java-Laufzeitsystems aus, durch die das Programm gestartet wurde

Information 7: Klasse Exception

Information 7 beschreibt zwei wichtige Methoden aus der von der Klasse Exception angebotenen Schnittstelle [MS+03].

Die Methode printStackTrace() liefert beispielsweise wichtige Informationen zum Inhalt des Laufzeitkellers, der zum Zeitpunkt bestand, als die Ausnahmesituation aufgetreten ist. Zu diesen Informationen gehört die Methodenaufrufkette, die in der Kurseinheit PROGRAMMIER-GRUNDLAGEN behandelt werden. Nähere Ausführungen zum Laufzeitkeller und der damit verbundenen Datenstruktur finden sich in der Kurseinheit OBJEKTORIENTIERTE PROGRAMMIERUNG.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

```
class CourseOverflowException extends Exception {     // inherits all attributes and
                                                      // methods from exception
  String overflowElement;                             // additional attribute
  CourseOverflowException (String shortName) {        // constructor
    overflowElement = shortName;
  }
}
```

- Definition einer eigenen Ausnahmeklasse
  - als Unterklasse von Exception
    - im Beispiel: class CourseOverflowException
- Ergänzung um zusätzliche Attribute und/oder Methoden, um den Fehler geeignet behandeln zu können

Information 8: Selbstdefinierte Ausnahmen

Es lassen sich durch Verfeinerung der Klasse Exception eigene Ausnahmeklassen definieren, wie das Beispiel in Information 8 verdeutlicht.

Die Ausnahme wird durch Erzeugung eines neuen CourseOverflowException-Objekts innerhalb einer throw-Anweisung ausgelöst. Bei der Erzeugung einer Ausnahme durch den Konstruktor CourseOverflowException() ist das Kursobjekt Course c, das den Überlauf verursacht hat, als Parameter zu übergeben.

1.4 Auslösen und Behandeln einer Ausnahme

Die throw-Anweisung ist dann auszuführen, wenn in der Methode addCourse(), die einen Kurs zum Kurskatalog hinzufügen soll, ein Überlauf des Katalogs festgestellt wurde.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Die Ausnahme wird durch die throw-Anweisung ausgelöst, wenn ein Kurs wegen Überlaufs nicht in den Katalog aufgenommen werden kann

```
public void addCourse(String shortName, int number) throws CourseOverflowException {
  if (numberEntries < entries.length)  // catalog not yet full?
    entries[numberEntries++] = new Course(shortName, number); // course added to catalog
 else  throw _____________________________________; // generation of exception
```

- Eine eventuell auftretende Ausnahme wird durch das try-catch-Konstrukt bei Aufruf und Ausführung der zur Klasse CourseCatalog gehörenden Methode addCourse() gefangen und behandelt

```
try { cc.addCourse("Info1", 1001);
} catch ( _____________________________ ) {
Out.println("CourseOverflowException caused by course " + _____________________);
// exception object and course title to be added
}
```

Interaktion 9: Auslösen und Behandlung einer Ausnahme

Im oberen Teil von Interaktion 9 ist die Methode, deren throw-Anweisung zu vervollständigen ist, angegeben.

Die Behandlung der von der Methode addCourse() ggf. erzeugten Ausnahme zeigt der im unteren Teil der Interaktion 9 angegebene Programmausschnitt. Der durch die try-Anweisung geschützte Bereich besteht hier nur aus einer Anweisung, dem Aufruf der Methode addCourse() zu einem Objekt cc der Klasse CourseCatalog. Sollte die (selbst definierte) Ausnahme CourseOverflowException auftreten, wird das mittels der obigen throw-Anweisung erzeugte Ausnahmeobjekt "eingefangen" und in diesem Fall durch eine Ausgabe behandelt. Die Ausnahmebehandlung ist geeignet zu ergänzen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- try-Anweisungen können über mehrere Methodendeklarationen hinweg verschachtelt sein
  - es werden bei den innersten catch-Klausel startend alle umgebenden catch-Klauseln überprüft, bis der passende Typ der erzeugten Ausnahme gefunden wurde

```
void m1() {
  try { ...
    m2();
} catch (E1 e) {
  // 3 ...
} catch (E2 e) { ... }
  // 4 ...
}
```

```
void m2() {
  ...
  m3();
  ...
}
```

```
void m3() {
  ...
  try { ...
    if (...)
      throw new E1(); // 1
    else
      throw new E2(); // 2
  } catch (E2 e) { // 5...}
  // 6 ...
}
```

Welcher Programmablauf entsteht, wenn in m3()

- Ausnahme E1
  // 1 // _____
- Ausnahme E2
  // 2 // _____

ausgelöst wird?

Interaktion 10: Suche eines Ausnahmebehandlers

Das Programmbeispiel in Interaktion 10 macht deutlich, dass try-catch-Konstrukte verschachtelt sein können. Die Suche nach einer zu dem aufgetretenen Ausnahmetyp passenden catch-Klausel erfolgt über die Verschachtelungen hinweg und endet, sobald die erste catch-Klausel gefunden wurde. Nach Ausführung der dazu gehörigen Anweisungsfolge wird das Programm nach der try-Anweisung, zu der die catch-Klausel gehört, fortgesetzt.

Zu den beiden im Programm in Interaktion 10 geworfenen Ausnahmen E1 und E2 soll der jeweils zugehörige Ausnahmebehandler gesucht und der resultierende Programmablauf soll beschrieben werden.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Das try-catch-Konstrukt kann durch eine finally-Klausel abgeschlossen werden
  - die Klausel wird immer (auch wenn gar keine Ausnahme aufgetreten ist) ausgeführt
  - dient dazu, die ggf. begonnenen Arbeiten im try-Block abzuschließen

```
try {
  ...
  ...
} catch (E1 e) {
  ...
} catch (E2 e) {
  ...
} finally {
  ...
}
```

- Welcher Ablauf ergibt sich, wenn im try-Block eine Ausnahme vom Typ E3 auftritt?

1. _____
2. _____
3. _____

Interaktion 11: finally-Klausel

Interaktion 11 geht auf eine spezielle Klausel, die finally-Klausel ein, die unabhängig davon, ob eine Ausnahme auftritt oder nicht, die Ausführung des try-catch-Konstrukts abschließt.

Im ausnahmefreien (Normal-) Fall würde im angegebenen Beispiel der vollständige try-Block und danach die finally-Klausel ausgeführt werden.
Entsprechend würde bei Auftreten einer der beiden Ausnahmen E1 bzw. E2 nach der entsprechenden catch-Klausel die finally-Klausel ausgeführt werden.
Der dritte Fall, das Auftreten einer Ausnahme, zu der keine catch-Klausel im try-catch-Konstrukt vorgesehen ist, wird im Rahmen von Interaktion 11 behandelt.

Mit den vorgestellten Konzepten zur Ausnahmebehandlung können alle Anforderungen, die weiter oben an ein Konzept einer effizienten und übersichtlichen Fehlerbehandlung gestellt wurden, erfüllt werden.

2 THREADS

Ein weiteres fortgeschrittenes und sehr mächtiges Programmierkonzept sind die *Threads*, durch die Teile eines Programms parallel ausgeführt werden können [Mö03].

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Ein Thread ist ein Programmstück, das parallel zu anderen Programmstücken abläuft
  - Programmstücke sind Prozesse, die parallel ablaufen
- Beispiel
  - Thread 1: Entgegennahme der Benutzereingabe
  - Thread 2: Durchführung von Berechnungen
  - Thread 3: Visualisierung von Zwischenergebnissen
- Parallelität bewirkt, dass sich die Threads nicht gegenseitig blockieren

Information 12: THREADS - Einführung

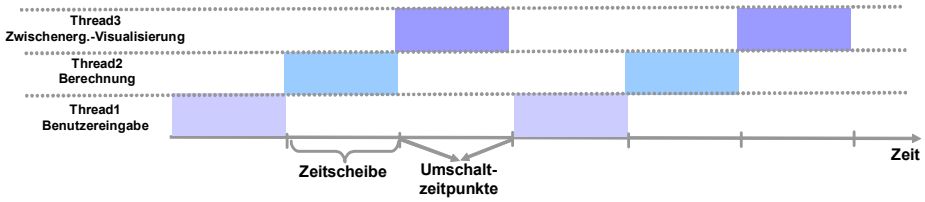
Die parallele Ausführung der in Information 12 genannten drei Beispiel-*Threads* bewirkt, dass eine Benutzereingabe (Thread 1) sofort entgegen genommen wird und nicht erst gewartet werden muss, bis beispielsweise eine angefangene Berechnung (Thread 2) beendet wurde oder ein Visualisierungsvorgang (Thread 3) abgeschlossen wurde.

2.1 Quasiparallelität

Das *Thread*-Konzept, durch das eine parallele Ausführung von Programmteilen ermöglicht wird, lässt sich auch auf Ein-Prozessor-Rechnern realisieren, was zu der so genannten Quasiparallelität führt.

- Echte Parallelität würde voraussetzen, dass mehrere Prozessoren zur Verfügung stehen

- Auf einem Rechner mit einem Prozessor werden die Threads quasiparallel ausgeführt
  - der Prozessor wird jedem der quasiparallel laufenden Threads nur eine gewisse (kurze) Zeitscheibe zugeordnet



Information 13: Quasiparallele Ausführung von *Threads*

Die quasiparallele Ausführung von *Threads* heißt, dass die beteiligten *Threads* den Prozessor nur jeweils eine gewisse Zeitscheibe zugewiesen bekommen.

Anhand der drei Beispiel-*Threads* wird die quasiparallele Verarbeitung in Information 13 veranschaulicht.


```
public class Thread {
  public void start() {...}
  public static void sleep(int milliSeconds) {...}
  public void run() {...}
  ...
}
```

- Threads sind in Java Objekte der Klasse Thread
  - bietet u.a. Methoden zum Starten start() und Verzögern sleep()
  - wichtigste Methode ist run()
    - enthält den Code, den der Thread im Laufe seines Lebens ausführen soll

- Implementierung eines Threads, indem eine Unterklasse zur Klasse Thread gebildet wird und run() mit dem gewünschten Code des Threads überschrieben wird

Information 14: Klasse Thread

In Java wird eine Klasse Thread bereitgestellt, die den Ausgangspunkt der Implementierung von *Threads* bildet. Ein Ausschnitt aus der Klasse Thread und das Vorgehen zur *Thread*-Implementierung sind in Information 14 gegeben.

```
class CharPrinter extends Thread {                  // thread CharPrinter is sub-class from Thread
  char signal;                                      // CharPrinter extends Thread by attribute signal

  public CharPrinter (char ch) { signal = ch;}      // constructor: gets a character ch from outside when called

  public void run() {                               // run() method overwritten by CharPrint
    for (int i = 0; i <= 20; i++) {
      Out.print(signal);
      int delay = (int) (Math.random() * 100);      // delay is a random number between 0  and 99
      try { sleep(delay); } catch (Exception e) { return; }     // thread sleeps for delay seconds
    }                                               // reason for try statement?
  }                                                 // _____
}
class ThreadCharPrinterProgram {
  public static void main (String[] arg) {
    CharPrinter thread1 = new CharPrinter('.'); CharPrinter thread2 = new CharPrinter('*');
    thread1.start(); thread2.start();               // generate and start thread1 and thread2
    Out.print('+');
  }
}
```

Interaktion 15: Beispiel-Programm zu *Threads*

In Interaktion 15 ist ein Beispiel-Programm angegeben, in dem ein *Thread* CharPrinter implementiert ist, der in zufälliger zeitlicher Abfolge ein Zeichen ausgibt. Welches Zeichen vom *Thread* auszugeben ist, wird bei dessen Erzeugung festgelegt.

Die zufällige zeitliche Abfolge wird durch die von der Klasse Thread geerbte sleep()-Methode und einer mittels der random()-Methode zufällig gewählten Wartezeit erreicht. Die Methode sleep() kann jederzeit durch die Benutzereingabe Strg-C abgebrochen werden, weshalb eine entsprechende Ausnahmebehandlung vorzusehen ist.

In der main()-Methode werden dann zwei dieser *Threads* erzeugt, die jeweils ein unterschiedliches Zeichen (. bzw. *) ausgeben. Nach dem Start der beiden *Threads* gibt das main()-Programm vor dessen Beendigung ein Zeichen + aus.

- Das Java-Laufzeitsystem erzeugt für jedes Programm einen neuen Thread
  - run()-Methode enthält die main()-Methode
- Jede run()-Methode hat ihre eigenen lokalen Variablen
- Threads sind in Java gewöhnliche Objekte, auf die andere Objekte gewöhnlich (über Zeiger auf dieses Thread-Objekt) zugreifen können
- Threads können ihrerseits auf andere Objekte zugreifen
  - Threads "teilen" sich diese Objekte, d.h. die Objektdaten werden nicht für jeden Thread kopiert
  - hierdurch ist die Kommunikation von Threads über die Objekt-Attribute möglich
  - erfordert allerdings ggf. einen synchronisierten Zugriff, um Inkonsistenzen auszuschließen

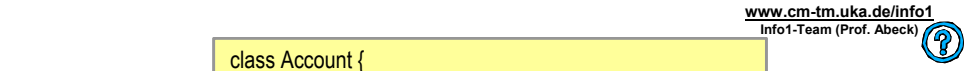
Information 16: *Thread*-Konzept in Java

Mit Hilfe des *Thread*-Konzepts kann jetzt auch geklärt werden, was beim Start eines Programms abläuft. Die für das main()-Programm innerhalb der run()-Methode angelegten Variablen sind lokal, d.h. der nächste gestartete *Thread* greift nicht auf diese Variablen zu, sondern erhält einen eigenen Speicherbereich für seine lokalen Variablen.

Da *Threads* ganz normale Objekte in Java sind, können andere Objekte auf diese zugreifen und umgekehrt können *Threads* auch auf andere Objekte zugreifen. Beim Zugriff auf andere Objekte ist allerdings zu beachten, dass unterschiedliche *Threads* auf ein und dasselbe Objekt zugreifen. Solche (quasi-) parallelen Zugriffe können zu Konflikten führen und erfordern daher eine im Folgenden behandelte Synchronisation der Zugriffe.

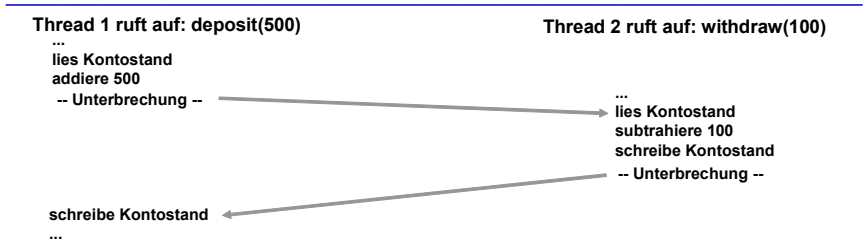
2.2 Synchronisation von *Threads*

An einem einfachen Beispiel soll zunächst erklärt werden, warum ein nicht abgestimmter Zugriff von mehreren *Threads* auf einen gemeinsamen Datenbestand zu unerwünschten Ergebnissen führen kann.



```
class Account {
  int balance;
  void deposit (int x) { balance = balance + x; }
  void withdraw (int y) { balance = balance - y; }
}
```

- Gegeben sei eine Klasse Account zum Verwalten (Einzahlen und Abheben) von Bankkonten
  - zwei Threads, von denen der eine auf das Bankkonto einzahlt (deposit()) und der andere vom selben Bankkonto abhebt (withdraw())
  - Wie lautet der Kontostand (balance) nach Beendigung des angegebenen Ablaufs?



Interaktion 17: Motivation für die Synchronisation von *Threads*

In dem in Interaktion 17 angegebenen Beispiel arbeiten zwei *Threads* auf einem gemeinsamen Objekt, einem Bankkonto (class Account). Während Thread 1 500 Geldeinheiten auf das Konto einzahlt, hebt Thread 2 parallel dazu 100 Geldeinheiten ab. Ein solcher Vorgang kann in der Praxis durchaus auftreten, wenn z.B. ein Kontostand "gleichzeitig" durch eine Gutschrift erhöht und durch eine (z.B. am Geldautomaten getätigte) Abhebung verringert wird.

Die Synchronisationsproblematik entsteht dadurch, dass ein Java-Befehl – im Beispiel die beiden Methodenaufrufe deposit() und withdraw() – vom Compiler in mehrere Schritte zerlegt wird und grundsätzlich nach jedem dieser Schritte eine Unterbrechung erfolgen kann. Die Unterbrechung führt dann dazu, dass ein anderer *Thread* die Bearbeitung aufnimmt.

Der beispielhafte Ablauf verdeutlicht, dass auf diese Weise Fehler auftreten können. Im konkreten Ablauf wird die Einzahlung der 500 Geldeinheiten nicht berücksichtigt.

- Kritischer Bereich ist der Bereich im Programm, in dem auf gemeinsame Daten zugegriffen wird
  - im Beispiel ist das der Bereich, in dem deposit() und withdraw() auf balance zugreifen
- Der Fehler entsteht dadurch, dass ein Thread unterbrochen wird, während er sich im kritischen Bereich befindet
- Lösung des Problems: Verzögerung der Unterbrechung, bis der Thread seinen kritischen Bereich verlassen hat
  - Einführung von Sperrvariablen

Information 18: Kritischer Bereich

Das Beispiel macht deutlich, dass es Bereiche im Ablauf eines *Threads* gibt, die nicht unterbrochen werden sollten, da sonst die Gefahr von Fehlern aufgrund nicht abgestimmter Zugriffe auf gemeinsame Daten besteht. Solche Programmbereiche werden als kritische Bereiche (siehe Information 18) bezeichnet.

Die Verhinderung einer Unterbrechung lässt sich durch so genannte Sperrvariablen erreichen.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- Sperrvariable ist ein beliebig erzeugtes dynamisches Objekt
  - vorzugsweise das Objekt, auf das nur synchronisiert zugegriffen werden darf

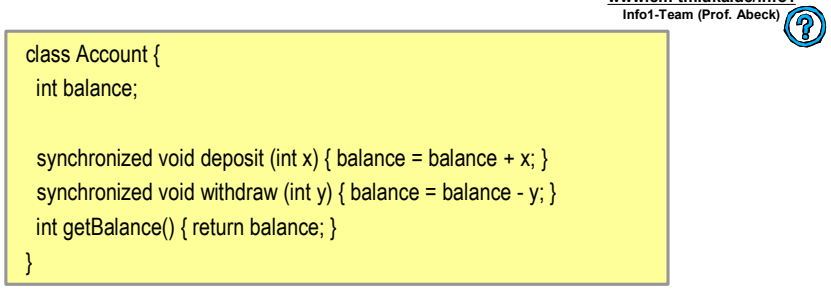
- Im Beispiel ist balance kein Objekt, sondern eine int-Zahl, weshalb ein Objekt lock eingeführt wird:

```
class Account {
  int balance;
  Object lock = new Object();

  void deposit (int x) { synchronized lock {balance = balance + x;} }
  void withdraw (int y) { synchronized lock {balance = balance - y;} }
}
```

Information 19: Sperrvariable in Java

Wie das Beispiel in Information 19 zeigt, kann in Java der Schutz des kritischen Bereichs durch die synchronized-Anweisung erfolgen, in der die Sperrvariable angegeben wird. Die synchronized-Anweisung bewirkt, dass vor Ausführung der darin enthaltenen Anweisungen – also vor Betreten des kritischen Bereichs – überprüft wird, ob bereits ein anderer *Thread* diese Sperrvariable gesetzt hat, sich also in einem durch die Sperrvariable geschützten kritischen Bereich befindet. Erst wenn sichergestellt ist, dass sich kein anderer *Thread* in einem durch die Sperrvariable geschützten Bereich befindet, erhält der *Thread* die Erlaubnis, den kritischen Bereich zu betreten. Solange er diesen Bereich nicht verlässt, werden alle diejenigen *Threads* gesperrt, die an eine entsprechende synchronized-Anweisung gelangen, in der die Sperrvariable angegeben ist.

Durch die synchronized-Anweisung wird somit erreicht, dass die Schreibzugriffe auf gemeinsam genutzte Variablen atomar, also nicht zerteilbar, durchgeführt werden.



```
class Account {
  int balance;

  synchronized void deposit (int x) { balance = balance + x; }
  synchronized void withdraw (int y) { balance = balance - y; }
  int getBalance() { return balance; }
}
```

- Java ermöglicht auch den Schutz ganzer Methoden durch die synchronized-Anweisung
- Eine Klasse, die Daten kapselt, auf die nicht gleichzeitig zugegriffen werden darf, wird auch als Monitor bezeichnet
- Erweiterung bzw. Aufweichung des Monitorkonzepts in Java
  - Methoden, die nur lesend auf die geschützten Daten zugreifen, müssen nicht als synchronized definiert werden (Beispiel: Methode getBalance())
  - Vorteil dieser Erweiterung:

Interaktion 20: Monitor

Anstelle der Einführung einer Variablen (im Beispiel lock), durch die mittels der synchronized-Anweisung eine Anweisungsfolge geschützt werden kann, ermöglicht Java auch den Schutz einer ganzen Methode. Das Beispiel in Interaktion 20 zeigt, wie das Problem des synchronisierten Zugriffs auf das Bankkonto mittels synchronized-Methoden gelöst werden kann. Werden in einer Klasse Methoden als synchronized deklariert, so wird während deren Ausführung sichergestellt, dass zu jedem Zeitpunkt immer nur höchstens eine dieser Methoden aktiv ist.

Eine Java-Klasse, in der alle Methoden als synchronized deklariert sind, wird in der Literatur auch als ein Monitor bezeichnet. In Java wird das Monitor-Konzept dahingehend erweitert bzw. aufgeweicht, dass Methoden, die nur lesend auf gemeinsame Daten zugreifen, nicht als synchronized definiert werden müssen. Das gilt allerdings nur für lesende Zugriffe, die atomar auf dem Rechner ausgeführt werden. Bei lesendem Zugriff auf Werte des Typs long oder double ist bei der heutigen Rechnergeneration daher ein Schutz mit synchronized erforderlich.

2.3 Deadlock

Beim Sperren von *Threads* muss darauf geachtet werden, dass nicht eine Situation eintritt, die als *Deadlock* oder Verklemmung bezeichnet wird.

www.cm-tm.uka.de/info1
Prof. Abeck & Info1-Team

- Ein Deadlock ist gegeben, wenn
  1. der sich im kritischen Bereich befindliche Thread erst dann weiterarbeiten kann, wenn eine bestimmte Bedingung erfüllt ist und
  2. diese Bedingung nur von einem der wartenden Threads erfüllt werden kann

- Beispiel
  - Das Konto darf nicht überzogen werden, d.h. withdraw(y) muss warten bis balance >= y
  - Die Bedingung kann nur durch Einzahlung auf das Konto erfüllt werden, aber deposit() ist gesperrt, weil sich withdraw() im kritischen Bereich befindet

Information 21: *Deadlock*

In einer *Deadlock*-Situation warten die *Threads* gegenseitig aufeinander: Der aktive *Thread* wartet darauf, dass einer oder mehrere der gesperrten *Threads* zur Erfüllung der Bedingung beiträgt und die gesperrten *Threads* auf den aktiven *Thread*, der aber nicht mehr weiterarbeiten kann.

Der *Deadlock* kann offensichtlich nur dadurch aufgehoben werden, dass der aktive *Thread* den kritischen Bereich freigibt und sich solange schlafen legt, bis sich die Bedingung erfüllt hat, die seine Weiterarbeit ermöglicht.

www.cm-tm.uka.de/info1
Info1-Team (Prof. Abeck)

- wait() bewirkt, dass der rufende Thread
  1. den Monitor freigibt
  2. sich schlafen legt

- notify() bewirkt, dass
  1. der auf die Erfüllung der Bedingung wartende Thread geweckt
  2. weiterarbeitet, wenn die Bedingung erfüllt ist
  3. wieder schlafen gelegt wird, wenn die Bedingung (noch) nicht erfüllt ist

- Beispiel:

```
synchronized void deposit (int x)
  { balance = balance + x;
  notify();
}
```

```
synchronized void withdraw (int y)
  try {
    while (balance < y) wait();
    // balance <= y
  } catch (InterruptedException e) {
    return;
  }
  // balance >= y
  balance = balance - y;
}
```

Information 22: wait und notify

In Java stehen zur Vermeidung eines *Deadlocks* die beiden zur Klasse Object() gehörenden Methoden wait() und notify() zur Verfügung. Da die Klasse Account (wie alle anderen Klassen) von Object() abgeleitet ist, können diese Methoden innerhalb von deposit() und withdraw() in der in Information 22 angegebenen Form genutzt werden.

Mit den *Threads* und den Sprachelementen zu deren Synchronisation wurden die wichtigsten Sprachelemente, die zur Programmierung von parallelen Algorithmen zwingend erforderlich sind, einführend vorgestellt. Gleiches gilt für die Ausnahmebehandlung und die Sprachelemente, die im Zusammenhang mit der Klasse Exception stehen.

Die vorliegende Kurseinheit könnte um einige weitere fortgeschrittene Programmierkonzepte erweitert werden. Genannt seien exemplarisch Konzepte zur Realisierung von Programmen, die auf der Grundlage moderner Web-Technologien verteilt über das Internet arbeiten [AL03, CJ02, KR03].

VERZEICHNISSE

Abkürzungen und Glossar

Abkürzung oder Begriff	Langbezeichnung und/oder Begriffserklärung
Ausnahme Klasse Exception	Objekte, die einen Ausnahme- oder Fehlerzustand signalisieren. Eine Ausnahme wird mit der throw-Anweisung ausgelöst und kann mittels eines catch/finally-Konstrukts abgefangen und behandelt werden [MS+03].
Ausnahmebehandler	Programmabschnitt, der nach Auftreten einer in einem geschützten Block aufgetretenen Ausnahme ausgeführt wird. Englischer Begriff: *Exception Handler*
Deadlock	Situation, in der Threads gegenseitig aufeinander warten. Deutscher Begriff: Verklemmung
Fehlercode	Festlegung (Codierung) der in einer Methode ggf. auftretenden Fehler in Form von Rückgabewerten, die an die aufrufende Methode zu übergeben sind.
Geschützter Block	Programmblock, der durch einen Ausnahmebehandler geschützt ist. Englischer Begriff: *Protected Block*
Kritischer Bereich	Programmbereich im Ablauf eines *Threads*, die nicht unterbrochen werden sollten, da sonst die Gefahr von Fehlern aufgrund nicht abgestimmter Zugriffe auf gemeinsame Daten besteht.
Thread	Ein parallel zu anderen Programmstücken ausgeführtes Programmstück.

Index

Informationen und Interaktionen

Literatur

[AL+03] Sebastian Abeck, Peter C. Lockemann, Jochen Schiller, Jochen Seitz: Verteilte Informationssysteme – Integration von Datenübertragungstechnik und Datenbanktechnik, dpunkt.verlag, 2003.

[CJ02] David A. Chappell, Tyler Jewell: Java Web Services – Using Java in Service-Oriented Architectures, O'Reilly & Associates, 2002.

[KR03] Jim Kurose, Keith Ross: Computer Networking – A Top-Down Approach Featuring the Internet, Addison-Wesley, Pearson Education, 2003.

[Mö03] Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.

[MS+03] Java – Programmierhandbuch und Referenz für die Java-2-Plattform, Standard Edition, dpunkt.verlag, 2003.