Inhaltsverzeichnis

24.1 Betriebssysteme 2
Übersicht 2
Windows 4
UNIX 5
24.2 Programmierung 8
Werkzeuge zur Programmentwicklung 8
Programmiermethodik 10
24.3 Programmiersprachen 21
Einführung in PASCAL 21
Einführung in C 21
Einführung in C++ 21
Einführung in Java 22
Einführung in Fortran 48
24.4 Informationen im Internet 59
Das World Wide Web 59
Browser 60
Nutzung der Internet-Dienste 61
Gezielte Informationssuche 65
HTML 67
24.5 Anwendungssysteme 71
Computeralgebra 71
numerische Werkzeuge 71

24.1 Betriebssysteme

Übersicht

a) Bedeutung und Klassifikation

Betriebssystem, Basissoftware eines jeden Computersystems. Wird bei jedem Rechnerstart aufgerufen und stellt fundamentale Funktionen wie das Ausführen von Programmen (Applikationen), Organisation des Arbeitsspeichers, Ausgabe auf dem Bildschirm, Eingabe über die Tastatur, Organisation des Dateisystems auf den Speichermedien oder Ansteuerung von Peripherie-Einheiten (Drucker, Modem usw.) zur Verfügung.
Single-Tasking-Betriebssystem, Betriebssystem, das die Ausführung nur eines Programms (Anwendung) auf einmal gestattet. Der Start einer zweiten Anwendung ist erst nach Beendigung der ersten möglich.

□ MS-DOS

Multi-Tasking-Betriebssystem, Betriebssystem, welches die gleichzeitige Ausführung mehrerer verschiedener Programme (Prozesse) ermöglicht.

□ UNIX, OS/2, MVS (IBM), VMS (Vax), Windows NT, Windows 95/98.

● Ein Multi-Tasking-Betriebssystem ordnet in der Regel jedem Prozeß eine bestimmte (kurze) Zeitspanne zu, in der der Computer diesen bearbeitet. Nach Ablauf dieser Zeit wird der nächste Prozeß behandelt, dann der darauffolgende, bis schließlich wieder der erste an der Reihe ist (Timesharing). Darüber hinaus können Prozesse verschiedene Prioritäten haben, so daß wichtige Prozesse, z.B. die Benutzeroberfläche, schnell auf Anforderungen reagieren können.

▷ Multi-Tasking stellt große Anforderungen an das Betriebssystem. So muß gewährleistet werden, daß jedem Prozeß ein eigener Bereich des Arbeitsspeichers zugeordnet wird, der von keinem anderen Prozeß überschrieben werden kann. Auch die Zuweisung der Zeitscheiben muß geregelt werden.

Single-User-Betriebssystem, Betriebssystem, welches nur die Benutzung durch einen Benutzer (zu einer Zeit) vorsieht. Es gibt Single-User-Betriebssysteme mit Single- und mit Multi-Tasking.

□ MS-DOS, OS/2, Windows 95

Multi-User-Betriebssystem, Betriebssystem, welches die gleichzeitige Benutzung desselben Computers durch verschiedene Benutzer vorsieht. Dies setzt natürlich Multi-Tasking voraus.

□ UNIX, MVS

▷ Ein Multi-User-Betriebssystem muß Benutzer durch Paßwörter identifizieren, Zugriffe auf Daten und Prozesse anderer Benutzer verhindern und die Systemressourcen gerecht verteilen.

□ Übersicht über die verschiedenen Kategorien von Betriebssystemen:

User	Task	Beispiele
Single	Single	MS-DOS
Single	Multi	OS/2, Windows 95/98
Multi	Multi	UNIX, MVS, VMS

▷ Windows NT ist ein “halbes Multi-User-System”: Es können nur nacheinander verschiedene Benutzer an einem Rechner arbeiten; die Daten verschiedener Benutzer sind aber gegeneinander geschützt.

b) Dateisystem

Dateisystem (**Filesystem**), vom Betriebssystem auf einem Speichermedium zur Speicherung von Daten (z.B. Texten, Bildern oder auch Programmen) bereitgestellter Bereich. Das Format zur Verwaltung der abgespeicherten Daten ist dabei vorgegeben. Bei allen UNIX- und Windows-Versionen sowie MS-DOS unterscheidet man:

- **Datei** (**File**), Menge von zusammengehörenden Daten, die zur Identifizierung mit einem eindeutigen Dateinamen gekennzeichnet sind. Die Daten werden meistens in der Form mehrerer Blöcke auf dem

Speichermedium verteilt. Zusätzliche Verwaltungsinformationen (z.B. Ort der Datenblöcke, Dateigröße, evtl. Zugriffsrechte, Erzeugungs- und Modifikationsdatum) liegen in einem extra dafür vorgesehenen Bereich.

- **Verzeichnis** (**Directory**), spezielle Art von Datei, die eine Liste von logisch zusammengehörenden Dateien enthält.

▷ Mit Hilfe von Verzeichnissen kann eine Vielzahl von Dateien strukturiert werden, z.B. nach Anwendungsgebieten oder Besitzern geordnet.

▷ Die physikalische Lage der Dateien auf dem Speichermedium ist unabhängig von der logischen Verzeichnisstruktur. Lediglich die Verweise auf die Dateien sind gemäß dieser Struktur abgespeichert.

▷ Ein Verzeichnis kann beliebig viele **Unterverzeichnisse** enthalten, die dieses wiederum unterteilen.

● Durch die Unterteilung von Verzeichnissen in Unterverzeichnisse ergibt sich eine Baumstruktur, der **Verzeichnisbaum**. Die Äste des Baumes sind die Verzeichnisse, die Knoten sind die Dateien.

Wurzelverzeichnis (**Rootverzeichnis**), Ausgangspunkt des Verzeichnisbaums. Einziges Verzeichnis, das nicht Unterverzeichnis eines anderen Verzeichnisses ist.
Elternverzeichnis, im Verzeichnisbaum das direkt übergeordnete Verzeichnis. Das Rootverzeichnis hat kein Elternverzeichnis.

● Jede Datei kann eindeutig identifiziert werden, indem man, vom Rootverzeichnis ausgehend, alle Unterverzeichnisse bis zur Datei selbst angibt. Diese Folge heißt **(absoluter) Pfad**.

▷ In UNIX werden die Pfadbestandteile durch `/` getrennt, in MS-DOS durch \.

□ UNIX: `/usr/bin/ls` , Rootverzeichnis: /
DOS: `\dos\graphik` , Rootverzeichnis: \

▷ Auf Kommandozeilen-Ebene merkt sich das System einen Pfad, das **aktuelle Verzeichnis**, in dem es bei fehlender Pfadangabe nach Dateien sucht. Man kann auch auf Dateien in übergeordneten Verzeichnissen einfach zugreifen, indem man das Elternverzeichnis durch `..` angibt. Damit erhält man einen **relativen Pfad**, der ab dem aktuellen Verzeichnis gilt.

□ absoluter Pfad: `\user\heiko\brief.txt`
dieselbe Datei, relativ zum Verzeichnis `\user\andrea`: `..\heiko\brief.txt`

▷ In Windows und MS-DOS sind die Verzeichnisbäume der verschiedenen Speichermedien (Platten, CD-ROM, Diskettenlaufwerke) getrennt. Sie werden durch einen vorangestellten Laufwerksbuchstaben unterschieden.
In UNIX gibt es nur einen großen Verzeichnisbaum, der sich über alle Dateisysteme erstreckt. Die Unterdateisysteme werden an selbst gewählten Stellen in den Verzeichnisbaum eingeklinkt (**gemountet**).

c) Benutzeroberfläche

Benutzeroberfläche, Schnittstelle zwischen Betriebssystem und Benutzer, die Ein- und Ausgaben abwickelt. Heute als **graphische Benutzeroberfläche** (**Graphical User Interface**, **GUI**), die einen möglichst intuitiven Umgang mit dem System ermöglichen soll. Eingaben kommen neben der Tastatur von der Maus, Ausgaben werden oft graphisch umgesetzt.
Fenster, in der Regel rechteckiger Bereich des Bildschirms, der Ein- und Ausgaben jeweils einer von mehreren, gleichzeitig laufenden Anwendungen (oder auch nur einen Aspekt einer Anwendung) zusammenfaßt.

● Fenster können sich überlappen, sogar verdecken. Immer genau eines ist **aktiv**, d.h. es reagiert auf Tastatureingaben.

□ Häufige Elemente zur Manipulation von Fenstern:

Element	Zweck
Rahmen	Vergrößern, Verkleinern
Titelzeile	Verschieben
Schaltflächen	auf maximale/minimale Größe bringen, Anwendung beenden

- ▷ Zu einer Benutzeroberfläche gehören meist eine Menge von Standard-Anwendungen, die das graphisch orientierte Arbeiten unterstützen.
- □ Datei-Verwaltung, Starten von Programmen, Taschenrechner, Uhr, Editor zum Erstellen von einfachen Texten.
- ▷ Eine besondere Anwendung ist die Kommandozeilen-Eingabe (**Shell**), die das text-orientierte Arbeiten mit Systembefehlen in einem eigenen Fenster ermöglicht.
- ▷ Zur Automatisierung nicht zu komplexer Aufgaben kann man eine Folge von System-Kommandos in eine Datei (**Shellskript** oder **Batchdatei**) schreiben und wie ein Programm aufrufen. Shells enthalten zu diesem Zweck Befehle zur Programmierung von Abfragen und Schleifen.

Windows

Windows, hier zusammenfassend für die PC-Betriebssysteme Windows 95/98 und Windows NT der Firma Microsoft.
Windows 95 und sein Nachfolger **Windows 98** sind Single-User-, Multi-Tasking-Betriebssysteme für den PC-Heimbereich. Sie umfassen neben den Basisfunktionen eine graphische Oberfläche und einige einfache Standard-Applikationen.
Windows NT ist ein Multi-Tasking-Betriebssystem für den Bürobereich mit eingeschränkten Multiuser-Funktionen wie Datei- und Prozeßschutzmechanismen. Es verwendet die gleiche Oberfläche wie Windows 95.

a) Die Windows-Oberfläche

- ▷ Wichtiger Bestandteil der Windows-Oberfläche ist die **Task-Leiste**, meistens am unteren Rand angebracht, die eine Liste der aktiven Anwendungen, einen Anzeigebereich für besondere Anwendungen und das Start-Menü enthält.
- ▷ Vom **Start-Menü** aus kann man alle installierten Anwendungen und Programme zur Systemkonfiguration aufrufen. Es enthält weiter eine Liste der zuletzt benutzten Dokumente, die direkt mit dem zugehörigen Bearbeitungsprogramm gestartet werden. Außerdem kann man von hier aus nach Dateien suchen, die Systemhilfe starten und Windows beenden.
- ▷ Häufig benutzte Programme oder Dokumente kann man auch in Form kleiner Bilder (**Icons**) auf dem Hintergrund (**Desktop**) ablegen und von dort aus starten.
- ▷ Wichtige Anwendung zur Datei-Verwaltung ist der **Explorer**. Er zeigt den Inhalt eines Verzeichnisses an sowie einen Bereich zur Darstellung der gesamten hierarchischen Verzeichnisstruktur. Mit der zunehmenden Bedeutung von Internet-Dokumenten ist er seit Windows 98 mit dem Internet-Explorer verschmolzen.

b) Die DOS-Kommandoeingabe

- ▷ Die Bedeutung der DOS-Kommandos hat aufgrund der graphischen Oberfläche stark abgenommen. Speziell zur Automatisierung mit Batch-Dateien oder bei schwerwiegenden Systemproblemen sind sie aber immer noch hilfreich.
- ● Syntax der DOS-Kommandos:

```
KOMMANDO  OPTIONEN  ARGUMENTE
```

Dabei ist `KOMMANDO` der Name des DOS-Befehls, `OPTIONEN` dienen der Modifikation des Befehls und `ARGUMENTE` sind die Objekte, auf denen das Kommando operiert. `OPTIONEN` haben meistens die Form `/X`, wobei `X` ein einzelner Buchstabe ist.

□ `dir /o /p c:\windows`

zeigt alle Dateien im Verzeichnis `c:\windows` an, und zwar seitenweise (`/p`) und in alphabetischer Reihenfolge (`/o`).

● Es wird nicht zwischen Groß- und Kleinbuchstaben unterschieden. Dateiennamen können zwar Groß- und Kleinbuchstaben enthalten, aber `klein.txt` und `KlEiN.TxT` bezeichnen dieselbe Datei.

□ Wichtige Kommandos:

Name	Funktion
`cd PFAD`	aktuelles Verzeichnis ändern
`dir PFAD`	Dateien im Verzeichnis anzeigen
`cp FILE1 FILE2`	Datei `FILE1` nach `FILE2` kopieren
`del FILE`	Datei `FILE` löschen
`ren FILE1 FILE2`	Datei `FILE1` in `FILE2` umnennen
`mkdir DIR`	Verzeichnis `DIR` anlegen
`rmdir DIR`	(leeres) Verzeichnis `DIR` löschen
`type FILE`	Inhalt der Datei `FILE` anzeigen
`more FILE`	Inhalt von `FILE` seitenweise anzeigen
`set VAR=VALUE`	Variable `VAR` auf Wert `VALUE` setzen
`echo %VAR%`	Wert der Variablen `VAR` anzeigen
`set`	Werte aller Systemvariablen anzeigen
`%1`, .. `%9`	Aufrufparameter einer Batchdatei
`if`, `for`, `goto`	Ablaufsteuerung in einer Batchdatei
`rem`	Kommentarzeile

● Viele DOS-Kommandos verwenden die Zeichen `*` bzw. `?` als Platzhalter (**Wildcards**) für beliebig viele bzw. genau ein Zeichen.

□ `ren *.txt *.bak`

nennt alle `.txt`-Dateien in entsprechende `.bak`-Dateien um.

● Ausgaben auf den Bildschirm (genauer: die **Standardausgabe**) können mit $>$ in eine Datei umgeleitet werden. Entsprechend kann mit $<$ eine Eingabe statt von der Tastatur aus einer Datei gelesen werden. Mit Hilfe des |-Symbols schließlich kann die Ausgabe eines Programms zur Eingabe eines anderen werden.

□
```
dir *.exe > liste.txt
type liste.txt | more
```

UNIX

Mit **UNIX** bezeichnet man eine Klasse von Multitasking-/Multiuser-Betriebssystemen, die eine gemeinsame Wurzel haben, sich aber in einigen Details unterscheiden.

□ Solaris (Sun), HP-UX (Hewlett-Packard), AIX (IBM), IRIX (SGI), Linux (auf PCs).

▷ UNIX wird hauptsächlich von Workstation-Herstellern auf ihren eigenen Maschinen eingesetzt. Durch den Siegeszug von Linux, einer freien UNIX-Version, gewinnt es aber auch auf PCs zunehmend an Bedeutung.

a) UNIX-Oberflächen

▷ Zwar gibt es mit **X-Windows** (auch als **X11** bezeichnet) schon lange ein System für fensterbasierte, netzwerkweite graphische Ein- und Ausgabe unter UNIX. Aber die Details der Benutzeroberfläche

unterscheiden sich sehr stark zwischen den UNIX-Varianten.

▷ Mit der Entwicklung des **Common Desktop Environments** (**CDE**) entstand eine einheitliche Benutzer-Oberfläche, die für die meisten kommerziellen UNIX-Versionen verfügbar ist.

▷ Unter Linux gibt es mehrere, sehr verschieden aussehende Oberflächen. Weit verbreitet ist **KDE**, das wesentliche Elemente von CDE und Windows beinhaltet.

▷ Eine Besonderheit von KDE (im Vergleich zu Windows 95) ist die Möglichkeit, seine Anwendungs-Fenster in mehreren Bildschirmen (**Desktops**) zu organisieren, zwischen denen man hin und her schalten kann.

▷ Wichtiger Teil der KDE-Oberfläche ist die Applikationsleiste, normalerweise am unteren Bildrand, von der aus man die Standard-Anwendungen erreichen kann. Hier findet sich auch die Möglichkeit, zwischen den Desktops zu wechseln sowie einige Menüs zum Starten von Applikationen, zur Konfiguration der Oberfläche oder zur Systemverwaltung. Eine zweite Leiste, die die aktiven Anwendungen anzeigt, befindet sich am oberen Bildrand.

▷ Auch unter KDE kann man häufig benutzte Programme oder Dokumente als Icons auf dem Desktop ablegen und von dort aus starten.

▷ KDE enthält einige Standard-Applikationen, darunter Dateimanager, Editor, Taschenrechner, Mailprogramm und Hilfesystem.

b) UNIX-Shells

▷ Zur text-orientierten Eingabe von Kommandos gibt es auf UNIX-Systemen verschiedene Shells, die sich in Details der Bedienung oder der zugehörigen Programmiersprache unterscheiden.

□ Folgende Shells stehen auf nahezu allen UNIX-Systemen zur Verfügung:
Bourne-Shell (`sh`), C-Shell (`csh`), Korn-Shell (`ksh`), ,,Bourne-again"-Shell (`bash`).
Die bash ist unter LINUX die Standard-Shell.

● Syntax von UNIX-Kommandos:

```
KOMMANDO  OPTIONEN  ARGUMENTE
```

Dabei ist `KOMMANDO` der Name des Shell-Befehls oder Programms, `OPTIONEN` dienen der Modifikation des Befehls und `ARGUMENTE` sind die Objekte, auf denen das Kommando operiert. `OPTIONEN` haben meistens die Form `-X`, wobei `X` häufig ein einzelner Buchstabe ist.

□ `ls -l -t /home/peter`

zeigt alle Dateien im Verzeichnis `/home/peter`, und zwar in ausführlicher Darstellung (`-l`) und sortiert nach dem Zeitpunkt der letzten Änderung (`-t`).

● Es wird strikt zwischen Groß- und Kleinbuchstaben unterschieden. Insbesondere wird `LS` nicht als Kommando zum Auflisten der Dateien verstanden und die Dateien `Gross.txt` und `gross.txt` sind verschieden.

□ Wichtige Kommandos:

Name	Funktion
`cd PFAD`	aktuelles Verzeichnis ändern
`ls PFAD`	Dateien im Verzeichnis ausgeben
`cp FILE1 FILE2`	Datei `FILE1` nach `FILE2` kopieren
`rm FILE`	Datei `FILE` löschen
`mv FILE1 FILE2`	Datei `FILE1` in `FILE2` umnennen
`mkdir DIR`	Verzeichnis `DIR` anlegen
`rmdir DIR`	(leeres) Verzeichnis `DIR` löschen
`cat FILE`	Inhalt der Datei `FILE` anzeigen
`more FILE`	Inhalt von `FILE` seitenweise anzeigen
`VAR=VALUE`	Variable `VAR` auf Wert `VALUE` setzen
`echo $VAR`	Wert der Variablen `VAR` anzeigen
`set`	Werte aller Systemvariablen anzeigen
`$1, ..$9`	Aufrufparameter eines Shellskripts
`if, for, until, while`	Ablaufsteuerung in einer Batchdatei
`#`	Kommentarzeile

- Die *Shell* interpretiert beim Aufruf eines Kommandos die Zeichen `*` und `?` als Platzhalter (**Wildcards**) für beliebig viele oder genau ein Zeichen in Dateinamen. Damit funktioniert dieser Mechanismus für alle Befehle, aber nicht immer so wie unter MS-DOS.

□ `ls *ju*`

listet alle Dateien im aktuellen Verzeichnis, die die Zeichenfolge `ju` enthalten (auch `juhu` oder `maju`)

□ `mv *.txt *.bak`

nennt *nicht* alle Dateien `*.txt` in entsprechende Dateien `*.bak` um, vielmehr geschieht folgendes:

Es gebe die Dateien `a.txt`, `b.txt`, `a.bak` und das Verzeichnis `juhu.bak`. Die Shell erzeugt dann die Kommandozeile

```
mv a.txt b.txt a.bak juhu.bak
```

Das `mv`-Kommando verschiebt nun die Dateien `a.txt`, `b.txt`, `a.bak` ins `juhu.bak`-Verzeichnis.

- Wie in DOS können Ein- bzw. Ausgaben mit $<$ bzw. $>$ in eine Datei umgeleitet und mit | Eingabe bzw. Ausgabe zweier Programme verbunden werden.

□
```
ls *.html > liste
cat liste | more
```

24.2 Programmierung

Werkzeuge zur Programmentwicklung

a) Compiler

Compiler, Programm, das einen in einer Programmiersprache vorliegenden Text (**Quellcode**) in eine Abfolge von grundlegenden Befehlen übersetzt, die auf einem bestimmten Rechnertyp direkt ausgeführt werden kann (**Maschinensprache**, **Objectcode**). Die Ergebnisdatei ist ein **ablauffähiges Programm** (**Executable**).

- ☐ Auf einem Linux-Rechner sind Compiler für alle gängigen Programmiersprachen kostenlos (im Rahmen des GNU-Projekts) verfügbar, etwa der cc für C und C++ oder der f77 für FORTRAN77. Für Windows-Betriebssysteme sind Compiler kommerziell erhältlich. Es gibt allerdings auch dort Versionen der GNU-Compiler.
- ▷ Ein Executable ist nur auf einem Prozessor eines bestimmten Typs und einem Betriebssystemtyp ablauffähig.
- ▷ Ein Compiler erkennt Fehler im Programmtext, die sich auf die grammatischen Regeln einer Sprache beziehen, wie Schreibweise oder Anordnung von Befehlen oder richtige Verwendung von Variablen (**Syntaxfehler**).
- ▷ Ein weiterer Schritt beim Compilieren ist die **Optimierung**. Hier versucht der Compiler, den erzeugten Code so zu verändern, dass er schneller abläuft, ohne dass sich dabei die Ergebnisse der Berechnung ändern. Dazu führt er z.B. geschickt Zwischenergebnisse ein oder stellt die Reihenfolge von Befehlen um.
- ▷ Unabhängige Programmteile (Unterprogramme) können einzeln übersetzt werden. Das Zusammenfügen der Teile übernimmt dann ein spezielles Programm, der **Linker** oder **Loader**, der in der Regel vom Compiler selbst aufgerufen wird.
- ▷ Häufig benutzte Unterprogramme fasst man zu **Bibliotheken** zusammen, aus denen der Linker die benötigten Routinen heraussucht und dem Programm hinzufügt.
- ☐ Es gibt Bibliotheken mit Routinen zur Ein-/Ausgabe (read, write), für mathematische Operationen (sin, log) oder mit Graphikbefehlen (drawLine, setColor).

b) Interpreter

Interpreter, Programm, das in einer Textdatei enthaltene Kommandos einer Programmiersprache einliest (oft zeilenweise) und direkt ausführt.

- ☐ Traditionell werden Programme in BASIC von einem Interpreter ausgeführt. In der stark erweiterten Form des Visual-BASIC hat es auch heute noch eine große Bedeutung.
- ☐ Interpretersprachen haben weite Verbreitung im Bereich der Internet/WWW-Programmierung und der Gestaltung von graphischen Benutzeroberflächen. Zu den wichtigsten gehören Javascript, Perl, Tcl/Tk.
- ▷ Da der Compilierungsprozeß wegfällt, sind Interpreter besonders nützlich in der frühen Phase einer Programmentwicklung, wenn sehr häufige Tests nötig sind (**Rapid Prototyping**). Außerdem sind die Programme ohne weitere Anpassung auf allen Rechnertypen ablauffähig (**portabel**).
- ▷ Interpreter sind wesentlich langsamer als compilierte Programme, da bei der Ausführung jeder einzelne Befehl neu übersetzt werden muß. Grundsätzlich können aber auch für Interpretersprachen Compiler geschrieben werden (etwa für Visual-BASIC), die das "endgültige" Programm dann in Maschinencode übersetzen.
- ▷ Eine Zwitterstellung zwischen Compiler- und Interpretersprache nimmt Java ein: Java-Programme werden zunächst in einen maschinenunabhängigen Zwischencode (**Bytecode**) übersetzt, der dann von einem Interpreter ausgeführt wird. Damit hofft man, Portabilität des Codes und hohe Geschwindigkeit zu vereinen. Inzwischen gibt es allerdings auch schon Compiler zur Übersetzung des Bytecodes.

c) Debugger

Debugger, Programm zur Untersuchung von Laufzeitfehlern eines anderen Programms.

- ▷ Ein Compiler entdeckt nur die Syntaxfehler in einem Programm, die entscheidenden Fehler werden erst beim Programmablauf sichtbar.
- ▷ Ein Debugger ermöglicht es, ein Programm schrittweise oder bis zum Auftreten bestimmter Bedingungen ablaufen zu lassen. Außerdem erlaubt er das Betrachten von Variableninhalten während des Ablaufs.
- ▷ Damit ein Debugger den Maschinencode und den Quellcode in Verbindung bringen kann, muß der Compiler Zusatzinformationen im Maschinencode unterbringen und darf keine Optimierungen vornehmen. Dafür sind spezielle Einstellung beim Übersetzungsschritt nötig.
- □ Bei UNIX-Compilern wird dazu meistens die Option `-g` verwendet:

```
cc -g meinprogramm.c
```

d) Profiler

Profiler, Programm, das Informationen über das zeitliche Verhalten eines Programms und seiner Teile ermittelt und darstellt.

- ▷ Insbesondere bei numerischen Anwendungen muss ein Programm nicht nur fehlerfrei, sondern auch möglichst schnell ablaufen. Ein Profiler hilft dabei, die Stellen zu finden, die bei der Ausführung besonders viel Zeit brauchen.
- □ Ein Profiler liefert etwa folgende Daten:
  - Zahl der Aufrufe einer Routine
  - verbrauchte Zeit für einen bzw. für alle Aufrufe einer Routine
  - Zeitverhalten anderer Programmteile (meistens Schleifen)
- ▷ Typische Arbeitsweise eines Profilers: Zunächst fügt der Profiler (oder auch der Compiler) Messroutinen in den Code ein. Beim Programmablauf werden die Zeiten und anderen Daten ermittelt und in eine Datei geschrieben. Anschließend werden diese Daten ausgewertet und (oft graphisch) dargestellt.
- ▷ Grundsätzliches Problem: Alle Messungen benötigen selbst auch Zeit! Gute Profiler können diese Zusatzzeiten (den **Overhead**) abschätzen und entsprechend korrigieren.

e) Integrierte Entwicklungsumgebungen

- ▷ Zur Vereinfachung der Programmentwicklung gibt es, vor allem unter Windows-Betriebssystemen, spezielle Programme, die die häufig benötigten Werkzeuge unter einer graphischen Oberfläche zusammenfassen (**integrierte Entwicklungsumgebungen**).
- □ Neben den Grundwerkzeugen Editor, Compiler, Linker, Debugger und Profiler sind oft noch Werkzeuge zur Verwaltung größerer, aus vielen Quelldateien bestehender Projekte vorhanden. Außerdem unterstützt der Editor in der Regel die Syntax der Sprache und erlaubt ein gezieltes Aufsuchen und Vergleichen der verschiedenen Programm-Module (**Code-Browser**, **Klassen-Browser**).
- ▷ Unter UNIX wird zum Übersetzen größerer Projekte das Programm `make` verwendet. Nach dem Erstellen einer Beschreibungsdatei (**Makefile**), die die benötigten Module und ihre Abhängigkeiten definiert, werden dann nur noch die Teile neu übersetzt, die sich seit dem letzten Lauf verändert haben.
- ▷ Noch weitergehende Unterstützung bei der Programmentwicklung bieten komponentenbasierte Umgebungen (z.B. Visual-BASIC oder Java Beans). Sie stellen fertige Module etwa für graphische Oberflächenelemente oder für einen Datenbankzugriff zur Verfügung, die graphisch dargestellt werden und interaktiv zu größeren Programmen zusammengesetzt werden können. Im Extremfall können so rein graphisch - ohne eigenen Quellcode zu erstellen - komplexe Anwendungen zusammengestellt werden.

Programmiermethodik

a) Strukturierte Programmierung

Häufiger Gebrauch der `goto`-Anweisung, die einen Sprung an eine beliebige andere Stelle im Programm bewirkt, zur Steuerung des Ablaufs führte oft zu unverständlichem Programmtext ("Spaghetti-Code"). Um verständlichere, leichter zu modifizierende Programme zu erhalten, wurden folgende Mechanismen eingeführt (zusammengefaßt unter dem Begriff der **strukturierten Programmierung**):

- Kontrollstrukturen, die die Benutzung des `goto` überflüssig machen und den Programmablauf deutlich werden lassen,
- Datenstrukturen, die den mit den Daten durchzuführenden Operationen entsprechen,
- Techniken zur Zerlegung der Programme in kleine Einheiten ("Unterprogramme"), die weitgehend unabhängig voneinander entwickelt werden können,

▷ Diese Ideen wurden konsequent umgesetzt in der Entwicklung der Programmiersprache PASCAL.

Kontrollstrukturen

Anweisung, Grundeinheit der Programmausführung.

□ Einfache Anweisungen sind z.B.

- Zuweisung eines Wertes an eine Variable,
- Ausführung von Basis-Operationen an Variablen,
- Ausführung eines Unterprogramms,
- Kombinationen davon.

▷ Was als "elementare" Operation gewertet wird, hängt von der Programmiersprache ab, in Assembler etwa eine Addition von ganzen Zahlen, in Fortran eine Sinus-Berechnung, in SQL eine Datenbank-Abfrage.

Block, Folge von Anweisungen, die als eine einzige Anweisung betrachtet wird.

▷ Ein Block wird in PASCAL durch `BEGIN`/`END` eingeschlossen, in C, C++ und Java in { und }.

Bedingte Anweisung, Anweisung, die nur bei Eintreten einer Bedingung ausgeführt wird.

● Grundsätzliche Form der bedingten Anweisung:

```
if Bedingung then
   Anweisung1
 else
   Anweisung2
```

Ist die Bedingung erfüllt, wird `Anweisung1` ausgeführt, sonst `Anweisung2`.

▷ Der `else`-Zweig kann entfallen. Andererseits können durch Schachtelung von `if`/`else`-Anweisungen mehrfache Fallunterscheidungen dargestellt werden.

▷ Zur Vereinfachung solcher Fallunterscheidungen dient die **Verzweigung**, die die Auswahl unter einer Reihe von Optionen ermöglicht. Sie wird mit `switch` (C, C++, Java), `case` (PASCAL) oder `select case` (Fortran) eingeleitet.

Schleife, wiederholte Ausführung von Anweisungen solange, bis eine Abbruchbedingung erreicht ist.

▷ Wird die Abbruchbedingung zu Beginn der Schleife geprüft, spricht man von **abweisender** oder **`while`-Schleife**, bei Prüfung am Ende der Schleife von einer **nichtabweisenden** oder **`repeat`-Schleife**.

▷ Die nichtabweisende Schleife wird immer mindestens einmal durchlaufen.

▷ Häufig benötigt man Schleifen, bei denen eine (ganzzahlige) Indexvariable einen festen Wertebereich durchläuft. Obwohl sich solche Schleifen leicht etwa mit Hilfe von `while`-Schleifen ausdrücken lassen, gibt es dafür eine spezielle Anweisung, die **Laufanweisung**, die auch als **`for`-** oder **`DO`-Schleife** bezeichnet wird.

Datenstrukturen

▷ Ein wichtiges Prinzip der strukturierten Programmierung ist die Zuordnung von Variablen zu **Datentypen**. Ein Datentyp legt den Wertebereich einer Variablen fest und gibt zugleich an, welche Grundoperationen damit ausgeführt werden können.

Einfache Datentypen, in einer Programmiersprache die grundlegenden vordefinierten Datentypen, vor allem zur Darstellung von Zahlen und Zeichen.

▷ In den meisten Sprachen sind Datentypen für ganze Zahlen und für Fließkommazahlen vordefiniert, oft in verschiedenen Versionen für unterschiedliche Wertebereiche. Dazu kommen Typen für Zeichen (Buchstaben, Ziffern, Satzzeichen usw.) und für Wahrheitswerte (wahr/falsch).

□ In C gibt es drei Typen für ganze Zahlen (`short`, `int`, `long`), die zunehmend größere Zahlen aufnehmen können. Allerdings ist der genaue Wertebereich nicht festgelegt, sondern systemabhängig.

▷ Wahrheitswerte werden auch als **logische** oder **boolesche Werte** bezeichnet. Entsprechende Datentypen sind `boolean` (PASCAL, Java) oder `logical` (Fortran). In C gibt es keinen solchen Typ.

Feld (**Array**), Datentyp, der mehrere Elemente des gleichen Typs zusammenfaßt, auf die mit Hilfe eines ganzzahligen Integerwertes zugegriffen werden kann.

▷ Die Elemente eines Feldes können auch von einem zusammengesetzten Typ (Struktur, Feld) sein. Handelt es sich insbesondere wieder um Felder, erhält man mehrdimensionale Felder.

▷ Ein- und mehrdimensionale Felder dienen vor allem bei numerischen Berechnungen zur Beschreibung mathematischer Vektoren und Matrizen, aber auch geometrischer Objekte wie regelmäßiger Gitter.

▷ Da Rechnungen mit Matrizen bei naturwissenschaftlich/technischen Anwendungen sehr häufig sind, gibt es in Fortran viele Operationen zur direkten Bearbeitung von Matrizen. Darüberhinaus gibt es Programme, die interaktive Matrix-Berechnungen ermöglichen (z.B. Matlab).

Zeichenkette (**String**), Folge von Zeichen, zur Abspeicherung von Texten.

▷ Zeichenketten sind in der Regel einfach Felder von Zeichen. Allerdings sind sie durch besondere syntaktische Vereinfachungen ausgezeichnet, vor allem die Angabe von konstanten Strings in (einfachen oder doppelten) Hochkommas.

▷ In Java werden Zeichenketten als Objekte der Klasse `String` realisiert. Dabei ist die interne Struktur eines Strings irrelevant und für den Programmierer unsichtbar.

▷ In PASCAL und Fortran ist die Länge eines Strings fest. In C und C++ ist sie variabel; dort wird das Ende einer Zeichenkette durch ein besonderes Zeichen (`'\0'`) angezeigt.

Struktur (**Record**), Datentyp zur Zusammenfassung von Elementen verschiedener Typen.

□ Eine Struktur könnte z.B. die in einer Datenbank gespeicherten Daten zu einer Person in einem Datentyp zusammenfassen, etwa den Namen als String, das Alter als Integer, die Namen der Kinder als Feld von Strings.

▷ Die einzige, generell vorkommende Operation mit einer Struktur ist der Zugriff auf die einzelnen Komponenten. Sie geschieht anders als beim Feld nicht durch einen Index, sondern durch einen Namen, der die jeweilige Komponente bezeichnet.

▷ In Java gibt es keinen eigenen Datentyp für Strukturen. Allerdings ist eine Klasse ohne Methoden, die also nur Datenfelder enthält, i.w. das gleiche wie eine Struktur.

Zeiger (**Pointer**), Datentyp für Größen, die auf andere Variable verweisen. Ihr Typ beinhaltet immer auch Information über den Datentyp, auf den gezeigt wird.

▷ Zeiger werden vor allem benötigt, um Datenstrukturen aufzubauen, deren Größe oder genaue Struktur erst bei Ablauf des Programms bekannt sind (dynamische Datenstrukturen, s. Abschnittc) .2).

□ Wichtige Operationen mit Zeigern sind:
- Zeiger auf ein vorhandenes Objekt verweisen lassen,
- Zeiger auf ein neu erzeugtes Objekt verweisen lassen,
- Aufsuchen des Objekts, auf das ein Zeiger verweist.

▷ C und C++ erlauben weitere Operationen wie das Erhöhen eines Zeigers um eins oder die Bildung der Differenz von Zeigern (**Zeigerarithmetik**) und erlauben damit einen direkten Zugriff auf den Speicher. Dies ist z.B. für die Programmierung von Hardware-Schnittstellen wichtig. Auf der anderen Seite ist es eine der Hauptursachen für schwierig zu findende Programmierfehler.

Unterprogramme

Unterprogramm, Teilprogramm zur Bearbeitung einer abgegrenzten Teilaufgabe, das mehrmals in einem Programm verwendet (“aufgerufen”) werden kann.

● Unterprogramme können geschachtelt werden (Unterunterunter...-Programme), wobei die erlaubte Schachtelungstiefe vom System (Betriebssystem und Compiler) abhängt.

● Zum Austausch von Informationen zwischen aufrufendem (Haupt-) und aufgerufenem Unterprogramm dient die **Parameterliste**, eine Liste von Variablen, denen bei jedem Aufruf eines Unterprogramms vom Hauptprogramm (ggf. unterschiedliche) Werte zugewiesen werden.

▷ Parameter können Informationen in drei Weisen weitergeben:
1. vom Haupt- zum Unterprogramm (Eingabedaten, das Unterprogramm liest),
2. vom Unter- zum Hauptprogramm (Ergebnisdaten, das Unterprogramm schreibt),
3. in beiden Richtungen (das Unterprogramm liest und schreibt).

Je nach Informationsrichtung können Parameter in verschiedener Weise an Unterprogramme übergeben werden:
call-by-value, Kopien der Parameterwerte werden an das Unterprogramm übergeben. Es kann die Werte im Hauptprogramm nicht ändern.
call-by-reference, das Unterprogramm erhält Verweise auf die Parameter und kann somit Werte an das aufrufende Programm zurückgeben.

▷ Es empfiehlt sich, für Eingabe-Parameter call-by-value, für gemischte und Ausgabe-Parameter call-by-reference zu verwenden. Bei der Übergabe großer Datenmengen (Feld, große Struktur) wird man aber häufig auch für Eingabe-Parameter call-by-reference wählen, um das zeitraubende Kopieren der Daten zu vermeiden. Allerdings muß man dann darauf achten, daß das Unterprogramm die Daten nicht verändert.

▷ Nicht alle Programmiersprachen erlauben dem Programmierer, für jeden Parameter frei zwischen beiden Übergabe-Methoden zu wählen.

□ In Fortran werden alle Parameter als Referenzen übergeben. Allerdings kann man sie als Eingabe-Parameter deklarieren, woraufhin der Compiler überprüft, daß sie im Unterprogramm nicht überschrieben werden.

□ In C wird nur call-by-value verwendet. Man kann jedoch auch einen Zeiger auf eine Variable übergeben, was effektiv call-by-reference entspricht.

● Unterprogramme können eigene Variable definieren (**lokale Variable**), die außerhalb des Unterprogramms nicht zu sehen sind und nach dem Beenden des Unterprogramms keinen definierten Wert mehr haben.

▷ Es gibt in einigen Sprachen die Möglichkeit, Variable in Unterprogrammen zu definieren, die ihren Wert zwischen verschiedenen Aufrufen eines Unterprogramms beibehalten (mit `static` in C, mit

`save` in Fortran). Dies sind keine lokalen Variablen im eigentlichen Sinn, auch wenn sie lokal definiert sind.

▷ In den meisten Sprachen kann man **globale Variable** definieren, auf die in allen Unterprogrammen direkt zugegriffen werden kann, ohne daß sie als Parameter übergeben wurden. Dies vereinfacht die Verbreitung von Information, die an vielen Stellen gebraucht wird. Es schafft aber auch schwer zu durchschauende Abhängigkeiten zwischen den Unterprogrammen.

Funktion, Unterprogramm, das einen ausgezeichneten Rückgabewert hat, der direkt in Ausdrücken weiterverwendet werden kann.

□ Mit Hilfe der Funktion `sin` kann man, wie in der mathematischen Schreibweise, einfach sagen:

```
a = 2*sin(x) + 3
```

statt mit einem Ausgabeparameter `ergebnis` schreiben zu müssen:

```
sin(x, ergebnis)
a = 2*ergebnis + 3
```

▷ In C werden Unterprogramme immer als Funktionen definiert, die allerdings auch den speziellen Typ `void` (d.h. gar nichts) zurückgeben können.

▷ Die Verwendung einer Funktion erweckt leicht den Eindruck, daß ihr Aufruf keine Auswirkungen auf das Programm hat außer der Berechnung des speziellen Wertes (wie bei mathematische Funktionen). Daher sollte man darauf achten, daß Funktionen nur Eingabe-Parameter haben und keine Nebeneffekte auslösen (etwa Ausgaben in eine Datei).

Rekursive Funktion, Funktion, die sich selbst aufruft. Dies kann auch indirekt geschehen (`a` ruft `b`, `b` ruft `a`).

▷ Damit keine unendliche Verschachtelung (und damit ein Programmabsturz) eintritt, muß eine rekursive Funktion ein Abbruchkriterium haben.

▷ Rekursive Funktionen sind besonders nützlich im Zusammenhang mit dynamischen Datenstrukturen. Gerade die üblichen Standard-Beispiele wie die Berechnung der Fakultät sind jedoch besser durch iterative Algorithmen zu lösen.

▷ Vor allem bei größeren Programmier-Projekten zeigten sich Schwächen der strukturierten Programmierung:

- Unterprogramme sind oft nicht voneinander unabhängig, weil globale Daten, aber vor allem gemeinsame Datenstrukturen, Abhängigkeiten zwischen ihnen bewirken.
- Die Vorgehensweise, ein Programm erst in Teilprobleme zu zerlegen, die später realisiert werden (**Top-down-Entwurf**), erzeugt stark problemspezifische Funktionen, die in anderen Projekten nicht wiederverwendet werden können.

b) Objektorientierte Programmierung

▷ Erfahrungen mit großen Programmprojekten führten zur Entwicklung neuer Programmiersprachen mit folgender Zielsetzung:

- Zerlegung des Programms in überschaubare Einheiten mit klarer Schnittstelle (“Objekte”)
- Trennung von Schnittstelle und Implementierung
- Wiederverwendbarkeit der Teile

Objekt-orientierte Programmiersprache, Programmiersprache, die diese Ziele durch geeignete Sprachmittel möglichst weitgehend erzwingt.

□ Beispiele für objekt-orientierte Programmiersprachen sind: Smalltalk, Eiffel, C++, Java und Erweiterungen älterer Programmiersprachen (Object-PASCAL, Object-COBOL).

▷ Neue Sprachmittel und anderer Sprachgebrauch sind Ausdruck einer geänderten Denkweise beim Programmieren. Grundidee ist eine stärkere Orientierung an der “realen” Welt: Ein Programm ist eine Ansammlung von miteinander wechselwirkenden “Objekten”.

▷ Der Erfolg der Idee der Wiederverwendbarkeit zeigt sich im Entstehen von großen Bibliotheken mit Standardelementen (**Klassenbibliotheken**).

□ Java und Smalltalk haben fertige Elemente zur Programmierung von graphischen Bedienungsoberflächen. C++ beinhaltet eine Vielzahl von “abstrakten Datentypen” wie Stack oder Menge.

● Die folgenden Grundprinzipien sind - bei unterschiedlicher Nomenklatur in den einzelnen Sprachen - für OOP charakteristisch:

- Datenkapselung
- Vererbung
- Polymorphie

Datenkapselung

Klasse, Datentyp, der Datenstrukturen und die diese manipulierenden Routinen zu einer Einheit zusammenfaßt.

Objekt, eine Variable eines Klassentyps; es hat einen **Zustand** (= Werte für die internen Datenstrukturen) und stellt **Methoden** (= Routinen, die das Objekt manipulieren) zur Verfügung.

▷ Neben diesen eher statischen (vom Programm ausgehenden) Definitionen steht der dynamische Sprachgebrauch der OOP: Ein Objekt ist gegeben durch sein Verhalten (Zustand und Methoden); eine Klasse ist die Zusammenfassung von gleichartigen Objekten.

▷ Weiterer Sprachgebrauch: Ein Objekt schickt einem anderen eine Nachricht (= ruft eine Methode des Objekts auf).

Schnittstelle, Menge aller von außen (= von einem anderen Objekt aus) sichtbaren Methoden und Daten.

● Alle nicht zur Schnittstelle gehörenden Elemente können geändert werden, ohne daß Programme, die diese Objekte nutzen, dies bemerken. Dies erlaubt völlig verschiedene Implementationen für dieselbe Klasse.

▷ Dadurch, daß interne Strukturen nur mit Hilfe vorgegebener Methoden manipuliert werden können, wird garantiert, daß Objekte nur sinnvolle Zustände annehmen.

● Kennzeichen von OOP ist ein häufig dynamisches Erzeugen von Objekten: Ein neues Objekt wird erzeugt, wenn es gebraucht wird.

▷ Kehrseite dieser Dynamik: Nicht mehr gebrauchte Objekte müssen beseitigt werden, damit genügend Speicher für die aktuellen Objekte zur Verfügung steht. Dafür gibt es zwei verschiedene Ansätze:

- explizites Löschen mit einer delete-Routine
  Vorteil: effizient
  Nachteil: schwierig zu programmieren, daher fehleranfällig
- automatisches Löschen durch das Laufzeitsystem (**Garbage Collector**)
  Vorteil: einfach
  Nachteil: oft langsamer

□ C++ nutzt delete-Funktionen (Effizienz steht im Vordergrund), Java und Smalltalk dagegen einen Garbage-Collector (Sicherheit geht vor).

□ Klasse Polygon
```
  öffentliche Methoden:
    neu(Seitenzahl, Punktliste)
    drehe(Winkel)
    verschiebe(2d-Vektor)
    fläche()
    schwerpunkt()
  intern:
    Anzahl der Punkte
    Liste der Punkte
```

▷ Hat man mit `neu` ein bestimmtes Polygon erzeugt, kann man es nur noch mit Hilfe von Drehungen und Verschiebungen verändern.

▷ Wird die Funktion `schwerpunkt` häufig gebraucht, kann man die Implementierung ändern, indem man den Schwerpunkt des Polygons in einer internen Variablen ablegt. Beim Erzeugen eines Polygons mit `neu` wird der Schwerpunkt gleich berechnet und von `drehe` und `verschiebe` entsprechend angepaßt, beim Aufruf von `schwerpunkt` nur noch dieser Wert ausgegeben. Der Benutzer der Klasse `Polygon` merkt von dieser Veränderung höchstens, daß sein Programm schneller geworden ist.

Vererbung

Vererbung, Prinzip der OOP. Es ist möglich, von bereits vorhandenen Klassen neue Klassen abzuleiten, die alle Eigenschaften (Methoden, Daten) der Basis-Klasse automatisch erhalten (erben), denen jedoch zusätzlich neue Datenstrukturen und Methoden zugewiesen werden können, die die Anwendung der Klasse über die der Basis-Klasse hinausgehen lassen. Methoden der Basis-Klasse können durch gleichnamige Methoden in der abgeleiteten ersetzt (**überladen**) werden.

▷ Die Nutzung der Vererbung ist in der Regel dann sinnvoll, wenn die abgeleitete Klasse sich als ein Spezialfall der Basisklasse interpretieren läßt (“ist ein”-Beziehung).

□ Klasse Dreieck erbt von Polygon
```
  öffentliche Methoden:
     Umkreisradius()
     Umkreismittelpunkt()
   intern:
     radius  (aus Performancegründen)
```

Hier ist die Verwandtschaft klar: Ein Dreieck “ist ein” Polygon.

▷ Eine Variable der Basisklasse kann ein Objekt einer abgeleiteten Klasse enthalten. Dies ist möglich, weil die abgeleitete Klasse ja alle Methoden der Basisklasse erhält und damit wie eine solche benutzt werden darf.

□ Eine Variable vom Typ Polygon kann ein Objekt der Klasse Dreieck beinhalten, denn: Ein Dreieck “ist ein “ Polygon.

● Auch bei Vererbung bleibt die Kapselung erhalten: Eine abgeleitete Klasse kann nicht auf die privaten Daten ihrer Basisklasse zugreifen. Allerdings gibt es in einigen Sprachen (z.B. Java, C++) die Möglichkeit, den Zugriff speziell für abgeleitete Klassen zu erlauben.

Mehrfach-Vererbung, eine Klasse kann von mehr als einer Basis-Klasse abgeleitet werden und erbt dadurch alle Methoden aller Basis-Klassen.

▷ Schwierigkeiten treten auf, wenn von zwei Basisklassen verschiedene Methoden mit gleichem Namen kommen. In einigen Sprachen (z.B. Java) ist Mehrfach-Vererbung daher grundsätzlich nicht möglich.

Polymorphie

□ Die Methode `fläche()` sei in `Polygon` und `Dreieck` verschieden implementiert: allgemeingültig in `Polygon`, einfacher und schneller in `Dreieck`.
Die Variable `bermuda` sei vom Typ `Polygon`, ihr sei aber ein konkretes `Dreieck` zugeordnet. Es stellt sich die Frage, welche Funktion benutzt wird, wenn man die Fläche von `bermuda` ausrechnen will: die spezielle oder die allgemeine? Antwort in OOPs: immer die spezielle.

● Wird eine Methode der Basisklasse in einer abgeleiteten Klasse neu definiert, so wird immer automatisch die spezielle Methode verwendet, auch wenn das abgeleitete Objekt einer Variablen der Basisklasse zugeordnet ist. Dies bezeichnet man als **Polymorphie**.

▷ Der Compiler kann der Variablen der Basisklasse u.U. gar nicht ansehen, ob ihr ein Objekt einer abgeleiteten Klasse zugewiesen werden wird. Daher wird erst zur Laufzeit des Programms festgelegt, welche Methode verwendet wird. Dieses Verfahren bezeichnet man als **late binding**.

abstrakte Basisklasse, Klasse, die Methoden definiert, ohne sie selbst zu implementieren.

● Von einer abstrakten Klasse dürfen keine Objekte erzeugt werden.

▷ Abstrakte Klassen sind sinnvoll, um ein einheitliches Verhalten für alle abgeleiteten Klassen festzulegen. Diese (oder sogar erst deren Nachfahren) müssen die Methoden ihrer abstrakten Basisklasse in die Tat (d.h. in Code) umsetzen.

□ Die abstrakte Klasse `Fläche` soll zweidimensionale Flächen beschreiben. Sie enthalte eine abstrakte Methode `fläche()`, die den Flächeninhalt bestimmt. Die davon abgeleiteten Klassen `Polygon` und `Kreis` sind nun gezwungen, `fläche()` zu implementieren (sonst sind sie selbst abstrakt).

c) Grundlegende Algorithmen

Algorithmus, Beschreibung eines allgemeinen Verfahrens zur Lösung eines gegebenen Problems mit Hilfe einfacher Schritte. Oft wird zusätzlich verlangt, daß ein Algorithmus nur endlich viele Schritte benötigt.

▷ Algorithmen können meist sehr allgemein beschrieben werden, sie sind von der konkreten Beschreibungs- oder Programmiersprache unabhängig.

▷ Zur Lösung häufig auftretender Probleme sind oft sehr gute (schnelle, speichereffiziente) Algorithmen entwickelt worden. Als Ergebnis langer Forschungsarbeit sind sie eigenen ad-hoc-Lösungen fast immer überlegen.

□ Wichtige Algorithmen aus verschiedenen Anwendungsgebieten:

- allgemein: Sortieren, Verwalten komplexer Datenstrukturen
- Numerik: LR-Zerlegung, schnelle Fourier-Transformation (FFT)
- Graphentheorie: aufspannender Baum

▷ Die folgenden Beispiele stellen naive Lösungen und wichtige Algorithmen gegenüber. Merke: Ein Auto kaufen führt meist schneller zum Ziel als das Rad neu erfinden.

Sortieren

▷ Sehr häufig müssen Daten sortiert werden, etwa um danach schneller auf sie zugreifen zu können. Im folgenden sei ein Array von ganzen Zahlen zum Sortieren vorgegeben.

□ Einfaches Verfahren: zunächst wird das kleinste Element gesucht und durch einen Tausch an die erste Stelle gebracht, dann das zweitkleinste an die zweite Stelle usw.:

```
void swap(int[] array, int i1, int i2) {
  int  tmp  = array[i1];
  array[i1] = array[i2];
  array[i2] = tmp;
}

void sort(int[] a) {
  for (int i=0; i < a.length-1; i++) {
    for (int j=i+1; j < a.length; j++) {
      if (a[j] < a[i] ) {
        swap(a, i, j);
      }
    }
  }
}
```

- Für ein Array mit n Elementen ist die Laufzeit dieses Verfahrens $\sim n^2$.

□ **quicksort**-Algorithmus: Die Elemente werden so in zwei (möglichst gleich große) Gruppen aufgeteilt, daß die Elemente der ersten Gruppe größer sind als die der zweiten. Beide Gruppen werden dann jeweils rekursiv für sich sortiert.

```
void quicksort(int[] a, int low, int high) {
  int i = low;
  int j = high;
  int x = a[(i+j)/2];   // "mittleres" Element
  do
    while (a[i] < x) {  // suche großes Element bei den Kleinen
      i++;
    }
    while (a[j] > x) {  // suche kleines Element bei den Großen
      j--;
    }
    if (i<=j) {
      swap(a, i, j);
      i++;
      j--;
    }
  while (i<=j);
  if (low<j) {  // Abbruchbedingung
    quicksort(a, low, j);
  }
  if (i<high) {  // Abbruchbedingung
    quicksort(a, i, high);
  }
}
```

- Wie schnell dieses Verfahren funktioniert, hängt davon ab, wie gut das “mittlere” Element `x` wirklich in der Mitte liegt. Eine genaue Untersuchung ergibt eine durchschnittliche Laufzeit $\sim n * log(n)$.

▷ Dieses Beispiel zeigt die Eleganz rekursiver Verfahren: Hat man erst einmal beide Gruppen getrennt, erfolgt deren Sortierung durch Rekursion. Wichtig sind die Abbruchbedingungen: Besteht eine Gruppe nur aus einem Element, ist man mit Sortieren fertig.

Dynamische Datenstrukturen

▷ Bei vielen Problemen ist die Anzahl und Verknüpfung der Daten am Anfang nicht bekannt oder ändert sich häufig. Eine primitive Möglichkeit zur Darstellung besteht darin, ein riesiges Array auf Vorrat

anzulegen und die Daten über zusätzliche Indizes zu verwalten - ein mangels besserer Datenstrukturen in FORTRAN77 häufig genutztes Verfahren.

Dynamische Datenstruktur, Datenstruktur mit während der Laufzeit des Programms veränderlicher Größe. Die Grundidee besteht darin, aus geeigneten Basiseinheiten (Knoten) veränderliche Strukturen zu bilden, indem man Knoten mit Zeigern zusammenhängt.

□ Je nach Anzahl von Zeigern pro Knoten und ihrer Verzweigungsart unterscheidet man u.a. **einfach verkette Listen** (ein Zeiger, der jeweils auf das nächste Element zeigt), **doppelt verkettete Listen** (zwei Zeiger pro Knoten, jeweils einen zum Vorgänger bzw. Nachfolger) und **Bäume** (mehrere Zeiger pro Knoten, keine geschlossenen Wege).

▷ Zur Unterstützung dynamischer Datenstrukturen braucht eine Sprache folgende Elemente:

- Zeiger bzw. Verweise
- Strukturen, um aus Daten und Zeigern Knoten zu bilden
- Funktionen zur Verwaltung von Speicher (`new`, `delete`)
- einen Wert für Zeiger, die auf nichts verweisen (`null`, `nil`), für Knoten ohne Nachfolger

□ Als einfaches Beispiel für einen Baum betrachten wir einen Familienstammbaum. Ein Knoten bestehe aus dem Namen einer Person sowie Zeigern auf Vater und Mutter. Die folgende Routine fügt den Namen der Mutter einer Person im Baum ein. Der Einfachheit halber gehen wir davon aus, dass alle Namen eindeutig sind.

```
class Knoten {
  String name;
  Knoten vater;
  Knoten mutter;

  Knoten(String meinName) {
    name   = meinName;
    vater  = null;
    mutter = null;
  }

  boolean einfuegeMutter(String nameMutter, String nameSohn) {
    boolean gefunden;

    // bin ich's ?
    if (name.equals(nameSohn)) {    // gefunden
      mutter = new Knoten(nameMutter);
      return true;
    }

    // oder ein Vorfahre mütterlicherseits ?
    if (mutter != null) {
      gefunden = mutter.einfuegeMutter(nameMutter, nameSohn);
      if (gefunden) {
        return true;
      }
    }
```

```
    // oder ein Vorfahre väterlicherseits ?
    if (vater != null) {
      gefunden = vater.einfuegeMutter(nameMutter, nameSohn);
      if (gefunden) {
        return true;
      }
    }

    // bei mir ist er nicht
    return false;
  }
}
```

▷ Der Fall, dass beim gesuchten Sohn schon eine Mutter eingetragen ist (vielleicht gar mit riesigem Stammbaum), wird hier rigoros abgehandelt: Durch die Zuordnung

```
mutter = new Knoten(nameMutter);
```

wird der Verweis auf die bisherige Mutter gelöst, ihr Knoten wird der Garbage-Collection übergeben.

▷ Operationen mit Bäumen sind die idealen Kandidaten für rekursive Algorithmen: Man beschäftigt sich nur mit dem eigenen Knoten und überläßt das Durchsuchen der gesamten Struktur der Rekursion.

Zerlegungs-Verfahren

▷ Bei verschiedenen Problemklassen führt folgende Grundidee zu sehr schnellen Algorithmen: Zerlege das Problem in kleinere Teilprobleme, die jeweils rekursiv weiter zerlegt werden, und fasse die Teile geschickt zum Gesamtergebnis zusammen. Solche Verfahren werden als **Zerlegungs-Verfahren** (**Divide and Conquer**) bezeichnet.

□ Wir betrachten die Multiplikation C = A*B zweier n x n - Matrizen. Das einfachste Verfahren setzt die Definition

$$C_{ij} = \sum_{k=1}^{n} A_{ik} * B_{kj}$$

direkt in Programmschleifen um.

● Anzahl der für die direkte Matrixmultiplikation erforderlichen Operationen:

Multiplikationen:	n^3
Additionen:	$n^3 - n^2$

□ Ein bekanntes Zerlegungs-Verfahren für die Matrixmultiplikation ist der **Strassen-Algorithmus**: Zunächst werden die Matrizen in 2x2 Blöcke von Teilmatrizen der Dimension $n/2$ zerlegt:

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Zur Berechnung der Teilmatrizen c_{ij} werden nun sieben Zwischenergebnisse m_i eingeführt und geschickt zusammengefaßt:

```
m1 = (a12 - a22)*(b21 + b22);
m2 = (a11 + a22)*(b11+b22);
m3 = (a11-a21)*(b11+b12);
m4 = (a11+a12)*b22;
m5 = a11*(b12-b22);
m6 = a22*(b21-b11);
m7 = (a21+a22)*b11;

c11 = m1 + m2 - m4 + m6;
c12 = m4 + m5;
c21 = m6 + m7;
c22 = m2 - m3 + m5 - m7;
```

Die dabei auftretenden Multiplikationen von Teilmatrizen werden rekursiv entsprechend berechnet.

▷ Ständiges Halbieren der Dimension funktioniert natürlich nur, wenn die Matrixdimension eine Zweierpotenz ist. Andere Dimensionen werden durch Einfügen von Nullen zur geeigneten Größe aufgebläht.

● Eine genaue Analyse der benötigten Rechenoperationen ergibt (falls n eine Zweierpotenz ist):

Multiplikationen: $n^{log_2 7} \approx n^{2.8}$
Additionen: $6 * (n^{log_2 7} - n^2)$

▷ Erst für sehr große Werte von n (Größenordnung 10000) überwiegt der gegenüber der einfachen Matrixmultiplikation kleinere Exponent den zusätzlichen Faktor 6. Ein gemischtes Verfahren, das zuerst mit Strassen-Halbierung arbeitet, aber ab einer bestimmten Größe die kleinen Teilmatrizen direkt multipliziert, ist schon bei Matrixgrößen unter 1000 effizient.

□ Eins der wichtigsten Zerlegungs-Verfahren ist die schnelle Fouriertransformation (**FFT**). Hier macht die gewaltige Beschleunigung gegenüber dem direkten Verfahren (n*log(n) statt n*n) viele praktische Fourieranalysen überhaupt erst möglich.

□ Verwandt ist auch die wichtige Klasse der **Mehrgitter-** oder **Multigrid-Verfahren** zur Lösung von partiellen Differentialgleichungen wie etwa der Poisson-Gleichung. Statt eine Lösung der gewünschten Genauigkeit direkt auf einem entsprechend feinen Gitter zu bestimmen, werden wiederholt Näherungen auf gröberen Gittern als Hilfsgrößen berechnet und geeignet auf feinere Gitter übertragen. Auch diese Verfahren liefern Beschleunigungen von n*log(n) gegenüber n*n.

24.3 Programmiersprachen

Einführung in PASCAL

Einführung in C

Einführung in C++

Einführung in Java

Java ist eine von der Firma Sun Microsystems entwickelte objektorientierte Programmiersprache. Ihre schnelle Verbreitung verdankt sie neben ihrer modernen, klaren Konzeption der Tatsache, daß sie die Erstellung kleiner Programmteile (**Applets**) ermöglicht, die direkt über das Netz geladen werden und in einem Browser-Fenster ablaufen. Damit wurde sie zur Programmiersprache des WWW schlechthin.

- Java-Programme werden in einen maschinenunabhängigen Zwischencode übersetzt, der von einem Interpreter der **virtuellen Java-Maschine** ausgeführt wird. Dieser kann auch in einen Browser integriert sein.
- ▷ **Java Development Kit** (JDK), grundlegende Entwicklungswerkzeuge wie Bytecode-Compiler zum Übersetzen von Java-Programmen in Bytecode, Interpreter zum Ausführen des Bytecodes und Appletviewer zum Anzeigen von Applets; können kostenlos von Sun bezogen werden.
- ▷ Java wird ständig weiterentwickelt, vor allem die Klassenbibliotheken enthalten neue Funktionalitäten. Problematisch: die Browser verwenden z.T. erst spät den neuen Stand. Die folgende Beschreibung basiert auf der Version Java 2 .

a) Grundlagen

Java lehnt sich in der Syntax stark an C++ an; insbesondere die Grundkonstruktionen finden sich hier wieder. Allerdings wurden viele problematische Punkte (z.B. Zeiger, Speicherverwaltung) nicht übernommen.

Programmaufbau

- Java-Programme benutzen keine vorgegebene Zeilenstruktur.
- Drei Formen von Kommentaren:

`//`	bis zum Zeilenende
`/* ... */`	beliebig über Zeilengrenzen, nicht schachtelbar
`/** ... */`	wird vom JDK-Programm javadoc zur Erstellung von Online-Dokumentation herausgezogen.

- Namen, die Variablen, Klassen etc. bezeichnen, dürfen aus Buchstaben, Ziffern und den Sonderzeichen _ und $ bestehen. Das erste Zeichen darf keine Ziffer sein.
- ▷ Java benutzt den internationalen Unicode-Zeichensatz; Buchstaben und Zahlen können also aus sehr vielen Schriften stammen. Darunter sind insbesondere die lateinischen Klein- und Großbuchstaben incl. aller Erweiterungen in europäischen Sprachen.
- □ Mögliche Bezeichner: `meinName`, `__intern__`, `Größe45`, `æΛć`
- Mindestens eine Klasse einer Java-Applikation (kein Applet) muß eine Funktion (**Methode**) `main` enthalten, die als `public static void` deklariert ist. Eine Java-Applikation wird gestartet, indem dem Interpreter der Name einer solchen Klasse angegeben wird.
- □ Das “Hello World”-Programm in Java, in der Datei Hello.java:

```
class Hello {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

Ausgabe mittels der println-Methode des Standard-Output-Streams.

Übersetzen:

```
javac Hello.java
```

Ausführen:

```
java Hello
```

Datenstrukturen

- In Java gibt es zwei Arten von Datentypen: einfache Typen und Referenztypen. Variablen der einfachen Typen speichern die entsprechenden Werte, Variablen von Referenztypen nur Referenzen (Zeiger) auf Objekte.

Einfache Datentypen, Darstellung ganzer oder Fließkomma-Zahlen, logischer Werte oder Zeichen, haben eine festgelegte Größe und interne Darstellung:

Datentyp	Bedeutung	Größe in Bit
`byte`	ganze Zahl	8
`short`	ganze Zahl	16
`int`	ganze Zahl	32
`long`	ganze Zahl	64
`float`	Fließkommazahl	32
`double`	Fließkommazahl	64
`boolean`	logischer Wert	1
`char`	Zeichen (im Unicode)	16

□ Konstanten dieser Typen:

Typ	Beispiele
`int`	`28`, `-12345`, `0x2a`
`long`	`1234567910L`
`float`	`3.1415926f`
`double`	`2.718281828`, `1.4e-45`
`boolean`	`true`, `false`
`char`	`'a'`, `'ö'`, `'\n'`

- ▷ Konstanten vom Typ `long` haben ein angehängtes `L` oder `l`, solche vom Typ `float` ein `F` oder `f`.
- ▷ Alle ganzzahligen Datentypen sind vorzeichenbehaftet.
- ▷ Fließkomma-Zahlen können auch die Werte $+\infty$ und $-\infty$ annehmen, mit denen "normal" gerechnet wird. Diese Konstanten werden in Java als Konstanten `Double.POSITIVE_INFINITY` etc. in den Klassen `Double` und `Float` definiert.
- ▷ `boolean` ist kein numerischer Typ, man kann keine arithmetischen Operationen mit ihm ausführen.
- Alle Variablen müssen vor dem ersten Auftreten deklariert werden, d.h. ihnen wird ein Typ zugewiesen. Diese Deklarationen können wie Anweisungen überall im Programm stehen; sie gelten bis zum Ende des sie umschließenden Blocks.
- ▷ Variablen eines numerischen Typs können solchen eines anderen numerischen Typs zugewiesen werden, wenn dadurch kein Genauigkeitsverlust möglich ist. Ansonsten muß man eine explizite Typumwandlung (**cast**) vornehmen.

□
```
int  i = 17;
long l = 128L;
l = i;              // ok, kein Verlust
i = l;              // Fehlermeldung vom Compiler !
i = (int) l;        // ok, wenn man weiss, was man tut
```

- Als höhere Datentypen gibt es Felder und Klassen. Variablen dieser Typen sind Referenzen, die auf Objekte zeigen. Die zugehörigen Objekte müssen mit **new** erzeugt werden.

▷ Objekte, die nicht mehr benötigt werden (d.h. auf die keine Variable mehr zeigt), müssen nicht explizit gelöscht werden. Stattdessen wird ihr Speicherplatz vom Java-Laufzeitsystem (genauer: dem **garbage collector**) automatisch freigegeben.

Feld (**Array**), Datentyp, der mehrere Elemente vom selben Typ zusammenfaßt.

▷ Die Zahl der Elemente eines Feldes `a` erhält man als `a.length`. Sie ist in Java *nicht* Teil des Typs, d.h. eine Array-Variable darf auf Arrays verschiedener Längen zeigen.

□
```
double[]  v1 = new double[10];
double[]  v2 = new double[20];
v1 = v2;         // ok, v1 zeigt jetzt auf v2
```

● Auf die Elemente eines Feldes `v` wird mit einem Index zugegriffen, der ein ganzzahliger Typ sein und zwischen `0` und `v.length - 1` liegen muß.

▷ Bei Index-Überschreitung wird eine Ausnahme vom Typ `IndexOutOfBoundsException` ausgelöst.

▷ Anfangswerte von Feldern durch Listen in geschweiften Klammern.

□
```
int[]  lotto = {2, 11, 12, 14, 34, 36};
for (int i=0; i < lotto.length; i++) {
  System.out.println(lotto[i]);
}
```

▷ Mehrdimensionale Felder sind Felder, deren Elemente selbst Felder sind.

□
```
double[]  matrix = new double[100][100];
```

Zeichenkette (**String**), Objekt der Standard-Klasse `String`.

▷ Besonderheiten für Strings:
- Konstante Strings: Zeichenfolgen, eingeschlossen in `"`
- Verketten von Strings mit `+`
- Methode `toString` zur Wandlung in einen String ist für alle Standardtypen definiert. Sie wird bei Bedarf vom Compiler eingefügt.

□
```
String  str = "Ergebnis: ";
double  result = 47.11;
System.out.println(str + result);
```

▷ Der Verbund-Datentyp (`RECORD` in PASCAL bzw. `struct` in C) zur Zusammenfügung von Elementen verschiedenen Typs ist in Java durch **class** ersetzt.

▷ Einen expliziten Zeiger-Datentyp gibt es nicht. Zwar enthalten alle Variablen der Referenztypen immer nur Zeiger, aber es gibt keine Zeigerarithmetik.

Operatoren

□ Die Operatoren sind C entlehnt. Die wichtigsten sind:
- arithmetische Operatoren +, - *, /, %
  Grundrechenarten und Modulo (Rest bei ganzzahliger Division)
- Vergleichsoperatoren ==, ! =, <, <=, >, >=
  Ergebnis ist vom Typ `boolean`
- logische Operatoren && (und), || (oder), ! (nicht)
  verknüpfen Werte vom Typ `boolean`

▷ Die C-typischen Abkürzungen gibt es auch hier:

```
i++      // für i=i+1
c *= 2   // für c = c * 2
```

▷ Division von ganzen Zahlen liefert den ganzzahligen Anteil ohne Rest.

● Beim Ausdruck `a && b` wird erst der Wert von `a` bestimmt. Ist dieser schon `false`, wird auch der ganze Ausdruck `false`, `b` wird dann nicht mehr berechnet. Analoges gilt für || und `true`.

▷ Dies ist wichtig, wenn die Berechnung von `b` einen Seiteneffekt hat.

▷ Für höhere Datentypen können keine Operatoren definiert werden (wie etwa in C++). Einzige Ausnahme: + zum Verketten von Strings.

▷ Die Gleichheit == (bzw. Ungleichheit !=) prüft bei Klassen und Feldern nur die Gleichheit der Zeiger, nicht die der einzelnen Komponenten. Dafür gibt es in allen Standardklassen die Methode `equals`.

Kontrollstrukturen

▷ Auch die Kontrollstrukturen wurden weitgehend C entlehnt.

Block, Folge von Anweisungen, in { und } eingeschlossen. Gilt als eine Anweisung.
Bedingte Anweisung:

```
if (boolescher Ausdruck)
  Anweisung1;
else
  Anweisung2;
```

Ist der boolesche Ausdruck `true`, wird `Anweisung1` ausgeführt, sonst `Anweisung2`.

▷ Der `else`-Teil kann entfallen.

▷ Hier wie im folgenden kann eine einzelne Anweisung durch einen Block ersetzt werden.

Verzweigung, Spezialfall von verschachtelten `if`-Anweisungen:

```
switch (Auswahl-Ausdruck) {
  case Konstante1:
     Anweisung1;
     break;
  case Konstante2:
     Anweisung2;
     break;
   ...
  default:
     AnweisungN;
}
```

Je nach Wert des Auswahl-Ausdrucks (ganzzahlig oder vom Typ `char`) wird der entsprechende `case`-Zweig angesprungen und die zugehörige Anweisung ausgeführt. Ist der Wert nicht aufgeführt, wird der `default`-Zweig benutzt. Beim `break` wird hinter die `switch`-Anweisung gesprungen.

▷ Fehlt ein `break`, so werden die Anweisungen des folgenden `case`- oder `default`-Zweigs ausgeführt. Dies kann manchmal sinnvoll eingesetzt werden, aber in der Regel ist es ein Fehler.

Abweisende Schleife:

```
while (boolescher Ausdruck)
    Anweisung;
```

Die Anweisung wird solange ausgeführt, bis der boolesche Ausdruck `false` ist. Ist er das schon zu Beginn, wird die Schleife übersprungen.

Nichtabweisende Schleife:

```
do
  Anweisung;
while (boolescher Ausdruck);
```

Wie die `while`-Schleife, aber der Test erfolgt erst am Ende, d.h. die Schleife wird immer mindestens einmal durchlaufen.

Laufanweisung:

```
for (Initialisierung; boolescher Ausdruck; Inkrementanweisung)
  Anweisung;
```

Dies ist eine abkürzende Schreibweise für

```
Initialisierung;
while (boolescher Ausdruck) {
  Anweisung;
  Inkrementanweisung;
}
```

□ Die `for`-Schleife ist zwar sehr allgemein einsetzbar, wird aber hauptsächlich für normale Zählschleifen benutzt:

```
for (int i=0; i<10; i++) {
  System.out.println(i + ", " + i*i);
}
```

break-Anweisung, dient zum sofortigen Verlassen einer Schleife oder einer Verzweigung.
continue-Anweisung, überspringt den Rest des aktuellen Schleifendurchgangs und beginnt mit dem nächsten (incl. Inkrementierung und Test).

□ `break` wird eingesetzt, wenn eine besondere Bedingung eintritt, die die weitere Schleifenausführung verhindert oder überflüssig macht:

```
for (int alter=18; alter <= 65; alter++) {
   boolean gewonnen = spieleLotto();
   if (gewonnen) break;
   arbeite();
}
```

▷ Im Fall von verschachtelten Schleifen beziehen sich `break` und `continue` normalerweise nur auf die sie direkt umgebende Schleife. Möchte man auch äußere Schleifen verlassen, muß man direkt vor den Beginn der entsprechenden Schleife eine Marke (**Label**) setzen und diese beim `break` bzw. `continue` mit angeben.

□
```
außen:
for (int i=0; i<n; i++) {
  for (int j=0; j<m; j++) {
    if (a[i][j] == 0) {
      break außen;
    }
    a[i][j] = 1/a[i][j];
  }
}
// hier landet man beim break
```

Behandlung von Ausnahmen

▷ Das Auftreten von unerwarteten Situationen beim Programmablauf wie fehlender Speicher oder Division durch 0 sollte durch geeignete Fehlerbehandlung aufgefangen werden, um zumindest ein völlig unkontrolliertes Programmverhalten zu verhindern. Java unterstützt dies durch besondere Mechanismen zur Ausnahmebehandlung.

Ausnahme (**Exception**), Objekt, das das Auftreten einer außerplanmäßigen Situation anzeigt. Sein Typ gibt Aufschluß über die Art des Problems.

□ Typische Ausnahmen sind:

Klasse	Bedeutung
`OutOfMemoryError`	kein Speicherplatz mehr bei einem new-Aufruf
`FileNotFoundException`	eine Datei des angegebenen Namens wurde nicht gefunden
`ArithmeticException`	Integer-Division durch Null
`IndexOutOfBoundsException`	Array-Index ist zu groß oder negativ

▷ Rechenoperationen mit floats oder doubles erzeugen keine Ausnahmen, sondern ggf. Werte wie unendlich (1.0/0.0) oder NaN (= Not a Number, etwa 0.0/0.0).

▷ Ausnahmen können sowohl ungewöhnliche “äußere” Umstände anzeigen als auch durch Programmierfehler verursachtes “inneres” Durcheinander.

● Tritt eine Ausnahme auf, sucht das Java-System nach einer umschließenden `try`-Anweisung, der eine `catch`-Anweisung für den richtigen Typ folgt. Diese `catch`-Anweisung wird dann ausgeführt und hinter evtl. weitere `catch`-Blöcke gesprungen. Wird kein passender `try`/`catch`-Block gefunden, wird das Programm mit einer Fehlermeldung abgebrochen.

```
try {
     // Anweisungen, die zu Ausnahmen führen können
}
catch (ExceptionTyp1 e) {
     // Anweisungen zur Behandlung von ExceptionTyp1
}
catch (ExceptionTyp2 e) {
     // Anweisungen zur Behandlung von ExceptionTyp2
}
  ...
```

▷ In einem `catch`-Zweig hat man zwei Möglichkeiten:

1. Fehler irgendwie beheben bzw. umgehen und weitermachen
2. Aufräumen und aufhören

▷ Man kann auch selber neue Typen von Ausnahmen definieren (etwa als von Exception abgeleitete Klasse) und sie im Problemfall mit Hilfe der **throw-Anweisung** auslösen. Eine Methode muß bei ihrer Deklaration anzeigen, daß sie solche Ausnahmen auslösen kann.

□
```
void dangerous() throws Exception {
  // ...
  if (error condition) {
     throw new Exception("problems in f");
  }
  // ...
}
```

▷ Wenn man die Routine `dangerous` aufruft, muß man durch einen geeigneten `try-catch`-Block sicherstellen, daß die Ausnahme auch behandelt wird. Dies prüft schon der Compiler und bricht an-

dernfalls mit einer Fehlermeldung ab.

▷ Insbesondere die Funktionen zur Ein- und Ausgabe mit Dateien lösen Ausnahmen aus, die abgefangen werden müssen. Damit wird sichergestellt, daß typische Fehler (Datei nicht vorhanden, nicht genug Plattenplatz etc.) berücksichtigt werden, zumindest in Form einer Fehlermeldung.

b) Objekte

▷ Java beinhaltet alle Konzepte der Objektorientierten Programmierung (s. Abschnitt b)), insbesondere
- Klassen
- Vererbung
- Polymorphie.

Klassen

Klasse, neuer Typ, der definiert wird durch Angabe seiner Komponenten und Methoden.
Objekt, Variable eines Klassentyps.

□
```
class Bruch {
  int zähler, nenner;           // Komponenten

  void  multipliziere(int n) {  // einzige Methode
    zähler *= n;
  }
}
```

▷ Komponenten können Variable beliebigen Typs sein, auch Arrays oder Klassen sind möglich.

● Methoden sind Funktionen, die Operationen mit den Komponenten ausführen können. Bei der Klassendefinition werden sie nicht nur mit Namen und Parameterliste angegeben (“deklariert”), sondern auch gleich vollständig als komplette Unterprogramme beschrieben (“definiert”).

▷ Methoden haben direkten Zugriff auf die Komponenten ihrer Klasse.

● Beim Aufruf einer Methode werden Kopien der aktuellen Parameterwerte angelegt, auf die innerhalb der Funktion zugegriffen wird (**call-by-value**).

▷ Für Variablen von einfachen Datentypen bedeutet dies, daß Änderungen daran sich nicht auf die aufrufende Funktion auswirken. Bei den höheren Datentypen ist der Wert einer Variablen aber immer ein Zeiger, über den auch in einer Funktion immer der Zugriff auf das Objekt selbst geschieht, nicht auf eine Kopie.

▷ Um auf die Komponenten und Methoden einer Klasse zugreifen zu können, muß es ein Objekt dieser Klasse geben (Ausnahme: statische Elemente).

● Die Deklaration einer Klassenvariablen erzeugt nur eine Referenz (einen Zeiger), aber noch kein Objekt. Dies geschieht erst durch die Anwendung des `new`-Operators.

□
```
Bruch p1 = new Bruch();
p1.multipliziere(3);
```

▷ Objekte müssen zwar erzeugt, aber nicht explizit vernichtet werden. Dies erledigt das Java-Laufzeitsystem selbst, sobald ein Objekt nicht mehr benutzt wird (**garbage collection**).

Konstruktor, spezielle Funktion mit dem Namen der Klasse, ohne Rückgabewert (auch nicht `void`). Er wird automatisch aufgerufen, wenn ein Objekt erzeugt wird.

▷ Die Aufgabe eines Konstruktor ist, den Komponenten sinnvolle Anfangswerte zuzuweisen. Komponenten, die nicht explizit initialisiert wurden, setzt der Compiler auf Null.

▷ Man kann mehrere Konstruktoren definieren, die sich in Anzahl bzw. Typ der Parameter unterscheiden. Dies ist auch für andere Methoden möglich, man bezeichnet es als **Überladen von Methoden**.

□
```
class Bruch {
  int zähler, nenner;   // Komponenten

  Bruch(int z, int n) {
    zähler = z;
    nenner = n;
  }

  Bruch(int n) {
    zähler = n;
    nenner = 1;
  }
}
```

● Innerhalb eines Konstruktors kann man - als allererstes - einen anderen, bereits definierten Konstruktor mit dem Namen **this** aufrufen.

▷ Dies ermöglicht, die Standard-Initialisierungen in einem Konstruktor unterzubringen, den man in allen anderen aufruft.

□ Im letzten Beispiel könnte man den zweiten Konstruktor also auch schreiben als

```
Bruch(int n) {
  this(n, 1);
}
```

● Wenn man *keinen* Konstruktor definiert, erzeugt der Compiler einen **Standard-Konstruktor**, der keine Argumente hat und alle Komponenten mit Null initialisiert.

▷ Das ist auch der Grund, warum man im Beispiel

```
Bruch p = new Bruch();
```

schreiben konnte, ohne einen Konstruktor zu definieren.

● Die Komponenten und Methoden, auf die von anderen Klassen aus zugegriffen werden darf, bilden die Schnittstelle einer Klasse. Sie werden durch das Attribut **public** gekennzeichnet. Mit **private** bezeichnete Komponenten sind von außen unsichtbar.

▷ Insbesondere können private Methoden und Komponenten ersetzt werden, ohne daß ein Benutzer einer Klasse etwas davon merkt, solange die Schnittstelle gleich bleibt. Dies erlaubt zum Beispiel, nachträglich die Interna einer Klasse zu ändern, um die Ausführungs-Geschwindigkeit zu erhöhen.

□
```
class Polygon {

  private Punkt[] points; // enthält auch Anzahl

  public Polygon(Punkt[] p) {...}
  public void drehe(double Winkel) {...}
  public double flächeninhalt() {...}
}
```

▷ Wird weder `public` noch `private` angegeben, ist der Zugriff innerhalb desselben Paketes erlaubt.

● Klassen können auch innerhalb anderer Klassen definiert werden. Diese **inneren Klassen** haben Zugriff auch auf die privaten Komponenten der umgebenden Klasse.

▷ Innere Klassen sind als Hilfsklassen der umgebenden Klasse nützlich, die "außen" nicht benötigt werden. Sie werden vor allem bei der Programmierung von graphischen Oberflächen eingesetzt (s. Applet-Beispiel im Abschnitt c) .5).

● Gelegentlich treten Größen auf, die nicht mit jedem einzelnen Objekt verknüpft sind, sondern mit der ganzen Klasse. Solche Komponenten werden **statisch** oder **Klassenvariable** genannt und mit dem Schlüsselwort `static` deklariert. Man kann auf sie zugreifen, indem man den Klassennamen verwendet.

□ Konstanten, die alle Objekte einer Klasse kennen oder die "global" bekannt sein sollen:

```
class Math {
  static double pi = 3.1415926;
  // ...
}

class Schätzer {
  double einfach(double daumen) {
    return Math.pi * daumen;
  }
}
```

□ Klassenvariable können auch dazu benutzt werden, die Zahl der erzeugten Objekte einer Klasse festzuhalten.

● Auch Methoden können statisch sein und ohne ein Objekt über den Klassennamen aufgerufen werden. Statische Methoden können nur auf statische Komponenten zugreifen.

□ Prominentes Beispiel: die statische Funktion `main`. Sie kann vom Java-System aufgerufen werden, noch bevor es ein Objekt gibt. Dies löst das alte "Henne-Ei-Problem" der objektorientierten Programmierung: Keine Objekte ohne erzeugende Funktion, keine Funktionen ohne umschließende Objekte.

Vererbung

▷ Wie im Abschnitt "Objektorientierte Programmierung" erläutert, erlaubt **Vererbung**, von einer vorhandenen Klasse neue abzuleiten. Die neue Klasse enthält dabei alle Komponenten und Methoden der Basisklasse.

● Das Schlüsselwort **extends** gibt an, daß eine Klasse von einer anderen abgeleitet wird.

□
```
class Dreieck extends Polygon {

  private double radius;
     // wird etwa aus Performancegründen zwischengespeichert

  public Dreieck(Punkt a, Punkt b, Punkt c) {...}

  public double umkreisradius() {
    return radius;
  }

  public Punkt umkreismittelpunkt() {...}
}
```

● Die Konstruktoren einer Klasse werden *nicht* an abgeleitete Klassen vererbt.

□ `Dreieck` enthält die privaten Komponenten `points` und `radius`, die öffentlichen Methoden `drehe`, `flächeninhalt`, `umkreisradius` und `umkreismittelpunkt` sowie den (öffentlichen) Konstruktor `Dreieck`. Es enthält aber keinen Konstruktor `Polygon`.

▷ Daß im obigen Beispiel alle Komponenten privat sind, ist durchaus typisch. Der Zugriff auf Komponenten sollte, falls überhaupt erforderlich, nur über öffentliche Methoden geschehen. Dies erlaubt eine saubere Trennung von Schnittstelle und Implementierung.

▷ Die abgeleitete Klasse enthält zwar die privaten Teile der Elternklasse, kann sie aber auch selber, d.h. in eigenen Methoden, nicht benutzen.

● Das Attribut **protected** erlaubt den Zugriff auf Komponenten und Methoden einer Klasse nur für abgeleitete Klassen.

▷ Erlaubt man abgeleiteten Klassen den Zugriff auf interne Komponenten, erleichtert man ihnen oft die Implementierung ihrer Methoden.

□ Die Klasse `Dreieck` des letzten Beispiels hat keinen Zugriff auf die Punkte `points`, daher wird es nicht gelingen, den Umkreismittelpunkt zu berechnen!
Mögliche Auswege:
- `Polygon` hat eine öffentliche Methode, die die Werte der Punkte zurückgibt. Gut gekapselt, aber ineffizient für `Dreieck`.
- Die Komponente `points` von `Polygon` wird `protected` statt `private`. Schnelle Implementierung, aber `Dreieck` sieht "Interna" von `Polygon`.

● Die erste Anweisung im Konstruktor einer abgeleiteten Klasse kann der Aufruf eines Konstruktors der Basisklasse sein. Er geschieht mit Hilfe des Schlüsselworts **super**.

□
```
class Dreieck extends Polygon {

  public Dreieck(Punkt a, Punkt b, Punkt c) {
    super( new Punkt[] {a, b, c} );
  }
  // ...
}
```

● Wird der `super`-Aufruf nicht verwendet, fügt der Compiler automatisch einen Aufruf `super()` ein. Er meldet einen Fehler, wenn es keinen solchen Konstruktor in der Basisklasse gibt.

▷ Dieser Fall tritt dann ein, wenn man in der Basisklasse eigene Konstruktoren definiert, aber keinen ohne Argumente.

Object, höchste Basisklasse, von der alle Klassen abstammen.

▷ Insbesondere sind alle Klassen, die nicht mit `extends` erben, direkt von `Object` abgeleitet.

▷ `Object` stellt einige Grundfunktionen zur Verfügung, z.B. die Funktionen `equals` und `toString`. Allerdings prüft `equals` nur auf Zeiger-Gleichheit und `toString` gibt einen internen Code aus. Daher sollte man diese Funktionen in eigenen Klassen überladen. In allen Java-Standard-Klassen ist dies geschehen.

● Um zu verhindern, daß eine Methode in einer abgeleiteten Klasse überschrieben wird, kann man sie als **final** kennzeichnen.

□ Man stelle sich eine Klasse vor, die Kontakt mit einem anderen Rechner vermitteln soll. Sie habe eine Methode `check`, die für die Paßwort-Übermittlung zuständig ist. Ist `check` nicht `final`, kann man eine neue Klasse ableiten, die `check` so umdefiniert, daß es das Paßwort gleich per Mail an einen Dritten weitergibt!

● Wird eine Klassen-Komponente als `final` gekennzeichnet, so kann sie nach der Initialisierung nicht mehr geändert werden, sie ist konstant.

▷ Für Klassen-Komponenten, die selbst Objekte sind, bedeutet dies, daß die Referenz sich nicht mehr ändern darf, d.h. sie zeigt immer auf dasselbe Objekt. Dieses Objekt ist dagegen nicht selbst konstant, seine Felder können sich ändern.

Polymorphie

▷ In Java kann jede Methode einer Klasse, die nicht `final` ist, in einer abgeleiteten Klasse überladen werden.

□
```
class Polygon {
  // ...
  public double flächeninhalt() {
    // kompliziertes Verfahren, allgemein die Fläche zu bestimmen
  }
}

class Dreieck extends Polygon {
  // ...
  public double flächeninhalt() {
    // einfache Flächenberechnung beim Dreieck
  }
}

class Dreiecktest {

  public static void main(String[] args) {
    Polygon[]   p = new Polygon[2];

    p[0] = new Dreieck();
    p[1] = new Polygon();

    for (int i=0; i<2; i++) {
      System.out.println("Fläche: " + p[i].flächeninhalt());
    }
  }
}
```

Für `i=0` wird die spezielle Methode `flächeninhalt` von `Dreieck` verwendet, obwohl der Zeiger `p[0]` selber vom Typ `Polygon` ist. Dies bezeichnet man als **Polymorphie** (s. Abschnitt b) .3).

▷ In der Regel kann erst zur Laufzeit entschieden werden, welche Methode für `flächeninhalt()` verwendet werden soll (“late binding”).

▷ Methoden, die `static` oder `private` sind, können nicht überladen werden, sie sind automatisch `final`.

▷ In C++ müssen Methoden, die von abgeleiteten Klassen überladen werden sollen, als `virtual` deklariert werden. Alle anderen Methoden können schon zur Übersetzungszeit gebunden werden. In Java sind alle (nicht-`final`) Methoden “virtual”.

● Eine Klasse kann Methoden enthalten, ohne sie zu implementieren. Solche Methoden werden mit dem Schlüsselwort **abstract** gekennzeichent, ebenso die ganze Klasse.

□
```
abstract class Fläche {
  // Komponenten und vollständige Methoden
  // ...
  abstract double flächeninhalt();
}
```

● Von einer abstrakten Klasse können keine Objekte angelegt werden.

● Klassen, die von einer abstrakten Klasse abgeleitet werden, können abstrakte Methoden implementieren. Werden alle Methoden implementiert, ist die abgeleitete Klasse nicht mehr abstrakt.

▷ Abstrakte Klassen dienen dazu, eine gemeinsame Schnittstelle für die von ihnen abgeleiteten Klassen zu erzwingen, selbst wenn noch nicht alle Methoden definiert werden können.

□ Alle "Flächen" (= von `Fläche` abgeleiteten Klassen) haben eine Methode `flächeninhalt()`.

Schnittstelle (**Interface**), Datentyp, der nur abstrakte Methoden und Konstanten enthält.

▷ Ein Interface entspricht einer abstrakten Klasse, die keine einzige Methode implementiert und auch keine Variablen (außer Konstanten) enthält.

□
```
interface Körper {
  void verschiebe(double x, double y, double z);
  void rotiere (double phi, double theta, double psi);
}
```

● Eine Klasse kann ein Interface implementieren, indem sie Definitionen für alle seine Methoden angibt. Sie zeigt dies durch das Schlüsselwort **implements** an. Dabei erbt sie auch alle Konstanten des Interfaces.

□
```
class Kugel implements Körper {
  double radius;
  // etc.

  void verschiebe(double x, double y, double z) {...}
  void rotiere (double phi, double theta, double psi) {...}
}
```

● Eine Klasse kann mehrere Schnittstellen implementieren.

▷ Mit Hilfe der Interfaces umgeht Java die Probleme der Mehrfachvererbung, die entstehen, wenn eine Klasse von zwei Basisklassen Methoden gleichen Namens erben soll.

c) Klassenbibliotheken

Wichtiger Bestandteil einer objektorientierten Sprache sind die mitgelieferten Klassenbibliotheken. Java bietet einen umfangreichen Satz für viele Anwendungsbereiche sowie einen Mechanismus zur Verwaltung und Identifizierung von Bibliotheken ("packages").

Paket (**package**), Menge von zusammengehörenden Klassen- und Interface-Definitionen.

● Der Name eines Pakets setzt sich aus mehreren Komponenten zusammen, die durch Punkt getrennt werden.

□ `java.lang`, `java.util.zip`, `javax.swing`

▷ Der Java-Interpreter sucht die Klassen eines Pakets in Unterverzeichnissen, die der Namenshierarchie folgen. Ausgangspunkt der Suche sind die Verzeichnisse, die in einer Umgebungsvariablen `CLASSPATH` angegeben werden.

□
```
CLASSPATH=C:\Programme\Java\Pakete;D:\extern
```

entsprechende Java-Klassen in

```
C:\Programme\Java\Pakete\java\lang
C:\Programme\Java\Pakete\java\util\zip
D:\extern\javax\swing
```

● Auf Elemente in Paketen kann immer durch Angabe des kompletten Namens zugegriffen werden. Die **import-Anweisung** erlaubt, zusätzlich auch abgekürzte Namen zu benutzen.

▷ Man kann bei der `import`-Anweisung ein Paket, eine Klasse eines Pakets oder alle Klassen eines Pakets angeben.

□ Zugriff auf statische Komponente `blue` der Klasse `Color` im Paket `java.awt`:

`import`	Zugriff auf `blue`
-	`java.awt.Color.blue`
`java.awt`	`awt.Color.blue`
`java.awt.Color`	`Color.blue`
`java.awt.*`	`Color.blue`

▷ Der * bezieht sich nur auf die Klassen des Pakets (im Beispiel `java.awt`), nicht auf mögliche Pakete in Unterverzeichnissen (etwa das Paket `java.awt.event`).

▷ Die Klassen des Pakets `java.lang` stehen automatisch direkt über die Klassennamen zur Verfügung.

● Alle Klassen eines Pakets haben Zugriff auf die nicht-privaten Komponenten und Methoden aller anderen Klassen des Pakets.

▷ Alle Felder, die keines der Attribute `private`, `protected` oder `public` tragen, stehen also genau innerhalb des Pakets zur Verfügung.

● Die Klassen einer Java-Quelldatei werden mit der **package-Anweisung**

```
package PAKETNAME;
```

einem Paket zugeordnet. Sie muß die erste Anweisung in einer Datei sein.

▷ Fehlt die `package`-Anweisung, gehören alle Klassen einer Datei zum unbenannten Standard-Paket. Dies ist für Tests und kleine Anwendungen völlig ausreichend.

▷ Die Standard-Java-Pakete sind in der Dokumentation des JDK ausführlich beschrieben , auf die man immer zurückgreifen sollte.

Grundlegende Klassen

Die folgenden Klassen stellen Basisfunktionen zur Verfügung. Man findet sie im Paket `java.lang`, das immer automatisch importiert wird.

String, Klasse zur Darstellung von konstanten Zeichenketten.

▷ Zur Erinnerung: Strings sind durch zwei syntaktische Besonderheiten ausgezeichnet:
- Bezeichnung von Strings mit Anführungszeichen `"`
- Verketten von Strings mit `+`

● Die Basisklasse `Object`, von der alle Klassen abgeleitet werden, enthält eine Methode `toString`, um ein Objekt als String darzustellen. Dies ist für alle Standard-Klassen in sinnvoller Weise definiert.

▷ Die Methode `toString` wird implizit aufgerufen, wenn Strings mit anderen Objekten verkettet werden.

□

Methoden von `String`	Bedeutung
`int length()`	Anzahl der Zeichen im String
`char charAt(int index)`	Zeichen an der Position `index`
`String substring(int begin, int end)`	Teilstring von `begin` bis `end - 1`
`int compareTo(String anotherString)`	lexikalischer Vergleich

▷ Strings sind *keine* einfachen Felder von Zeichenketten. Die interne Darstellung eines Strings ist - natürlich! - verborgen. Insbesondere ist von einem besonderen Endezeichen `\0` wie in C nichts zu sehen.

▷ Es gibt innerhalb der Klasse `String` keine Methode, einen String in eine Zahl umzuwandeln. Allerdings enthalten die “Wrapper”-Klassen `Integer`, `Long`, `Float` und `Double`, die jeweils eine Zahl in eine Klasse verpacken, dafür die Methode `valueOf`.

□
```
String  str1 = "3.1415926";
String  str2 = "Hanno" + "ver";
Double  object_pi;
double  pi;

object_pi = Double.valueOf(str1);
pi = object_pi.doubleValue();
str1 = str2.substring(1,4) + 96;  // implizites toString
```

StringBuffer, Klasse für veränderliche Strings

● Ein `StringBuffer` enthält einen internen Pufferspeicher, der bei Bedarf automatisch wächst.

▷ Es ist aufwendig, neuen Speicher zu besorgen. Man sollte daher versuchen, einen `StringBuffer` gleich mit der benötigten Größe zu initialisieren.

□

Methoden von `StringBuffer`	Bedeutung
`StringBuffer(int length)`	Konstruktor für einen `StringBuffer` mit Buffergröße `length`
`insert(int i, String str)`	fügt `str` an der Stelle `i` ein
`append(String str)`	hängt `str` an den `StringBuffer` an
`replace(int i1, int i2, String str)`	ersetzt die Zeichen von `i1` bis `i2 - 1` durch `str`
`delete(int i1, int i2)`	löscht Zeichen von `i1` bis `i2 -1`
`int length()`	Länge des `StringBuffers`

▷ Die Länge des `StringBuffers` ist die Zahl der Zeichen, nicht die Größe des internen Buffers.

□
```
StringBuffer str = new StringBuffer("Altglas");
str.replace(1,4, "Jung");     // str wächst auf 8 Zeichen
str.delete(4,5);                // str wird wieder kürzer
```

Math, Klasse, die die wichtigsten mathematischen Konstanten und Funktionen als statische Größen zusammenfaßt.

□

Felder von `Math`	Bedeutung
`Math.PI`, `Math.E`	π und e
`pow`, `exp`, `ln`, `sqrt`	Potenz, Exponentialfunktion, Logarithmus, Wurzel
`cos`, `sin`, `tan`, `asin`, ..	trigonometrische Funktionen und Inverse
`min`, `max`	Minimum, Maximum
`floor`, `ceil`, `round`	ganzzahlige Werte in der Nähe
`random`	Zufallszahl zwischen 0.0 und 1.0

▷ Da Java-Programme auf allen Rechnertypen identische Ergebnisse liefern sollen, sind die mathematischen Funktionen durch Bezug auf bestimmte Algorithmen genau festgelegt.

System, Klasse mit einigen betriebssystemnahen Funktionen. Sie enthält ebenfalls die Pointer auf Standard-Ein- und Ausgabe.

□

Felder von `System`	Bedeutung
`System.in`	Standard-Input als `InputStream`
`System.out`, `System.err`	Standard-Output- und -Fehler-Kanal als PrintStream
`long currentTimeMillis()`	Zeit seit 1.1. 1970 in Millisekunden
`String getProperty(String s)`	holt einige System-Informationen (CLASSPATH, Username, ...)
`void exit(int)`	beendet das laufende Programm

Ein- und Ausgabe

Das Paket `java.io` stellt eine Vielzahl von Klassen zur Verfügung, um größtmögliche Flexibilität zu erreichen. Die Ein-/Ausgabe über Tastatur und Bildschirm sowie mit Dateien kommt aber mit einigen Standard-Klassen aus.

Stream, Strom von Daten, die nacheinander (seriell) von einer Quelle eintreffen (**Input Stream**) bzw. an einen Empfänger gesendet werden (**Output Stream**).

☐ Streams können Verbindungen sein zu

- Tastatur/Bildschirm
- Datei
- String (Daten im Speicher)
- Netzwerk-Verbindung

▷ Das Stream-Konzept erlaubt, Daten auf einheitliche Weise von ganz verschiedenen Medien entgegenzunehmen bzw. weiterzuschicken.

● Es gibt zwei Sätze von abstrakten Basisklassen:

Reader/Writer bearbeiten Streams von Zeichen
verwenden 16bit-Unicode für internationale Nutzung
für formatierte Daten

InputStream/OutputStream bearbeiten 8bit-Zeichen (Byte Streams)
typisch für binäre Daten (Bilder, Sound)

● konkrete Klassen speziell für Dateien:

- `FileReader`/`FileWriter`,
- `FileInputStream`/`FileOutputStream`

☐

Methoden von `FileReader`/`FileWriter`	Bedeutung
`Konstruktor(String s)`	Erzeugen über einen Dateinamen, entspricht dem Öffnen der Datei
`int read(char[] buf)`	Einlesen von Zeichen gibt -1 zurück bei Dateiende
`void write(String s)`	Ausgeben eines Strings
`void write(char[] buf)`	Ausgeben eines Arrays von Zeichen
`void close()`	Schließen der Datei

Filter Streams oder **Processing Streams**, Klassen, die die Standard-I/O-Klassen um bestimmte Eigenschaften erweitern.

☐ **BufferedReader/BufferedWriter** Daten werden gepuffert und immer in ganzen Blöcken zu/von der Datei übertragen.

PrintWriter vereinfachte Ausgabe mit `print` und `println`, speziell für numerische Daten.

DataInputStream einfaches Einlesen nicht-formatierter Daten mit `readInt()`, `readDouble()` etc.

▷ Die Verwendung der Klassen mit Pufferung führt bei der Verarbeitung von Dateien zu großen Geschwindigkeitsgewinnen. Um eine gelegentlich notwendige unmittelbare Ausgabe zu erzwingen (etwa für Fehlermeldungen), kann man mit der Methode `flush()` den Puffer sofort leeren.

▷ `System.in` ist ein `InputStream`, `System.out` und `System.err` sind `PrintStream`s.

☐ Kopieren eines Files:

```
try {
  BufferedReader in  =
        new BufferedReader(new FileReader("juhu.in"));
  BufferedWriter out =
        new BufferedWriter(new FileWriter("juhu.out"));
  int c;         // enthält gelesenes Zeichen oder -1

  while ((c = in.read()) != -1){
    out.write(c);
  }
  in.close();
  out.close();
}
catch(IOException e) {
  System.out.println("Fehler beim Kopieren:");
  e.printStackTrace(System.out);
}
```

Collection-Klassen

Collection-Klasse, Klasse, die Elemente zu einer Einheit zusammenfaßt.

▷ Die Collection-Klassen sind im Paket `java.util` enthalten.

□ Wichtige Arten von Collection-Klassen:

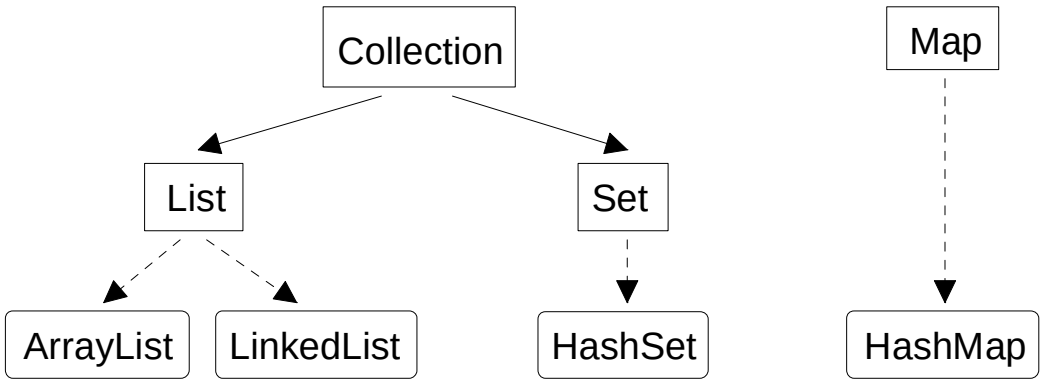
Menge ungeordnet, enthält kein Element doppelt,

Liste geordnet, Elemente möglicherweise mehrmals

Hashliste Paare von Schlüsseln und Werten, Schlüssel eindeutig

▷ Hashlisten heißen auch **Maps** oder **assoziative Arrays**.

□ Die wichtigsten Interfaces und Klassen:



▷ Die wichtigsten Methoden einer Liste (`List`) oder Menge (`Set`) stammen schon aus dem grundlegenden Interface `Collection`:

□

Methoden von `Collection`	Bedeutung
`boolean add(Object)`	fügt `Object` hinzu (in `Set` nur, falls es nicht schon drin ist)
`boolean remove(Object)`	entfernt `Object` (falls es drin ist)
`boolean contains(Object)`	prüft, ob `Object` enthalten ist
`int size()`	Anzahl von Elementen
`boolean isEmpty()`	prüft, ob überhaupt Elemente vorhanden sind

● Die Elemente der Collection-Klassen sind immer vom Basistyp `Object`.

- ▷ Dies erlaubt, Objekte beliebiger Klassen in einer `Collection` zu sammeln.
  Nachteil: Will man die Methoden eines Elements benutzen, muß man es mit einer expliziten Typumwandlung (Cast) wieder in seine alte Gestalt zurückverwandeln.
- ▷ Eine bessere Lösung bieten die Template-Klassen in C++: Sie enthalten den Typ der Elemente als Parameter.
- ● Collection-Klassen wachsen und schrumpfen bei Bedarf automatisch.
- ▷ Man sollte eine Collection-Klasse (außer bei der Erzeugung mit `new`) möglichst immer über das Interface ansprechen, nicht über die konkrete Klasse. Dies erlaubt, die konkrete Klasse (und damit die Implementierung) zu wechseln, falls es aus Performance-Gründen nötig ist.
- □ **LinkedList** implementiert eine doppelt verkettete Liste
  schnell beim Einfügen und Löschen von internen Elementen
  spezielle Methoden zur Bearbeitung des ersten und letzten Elements

  ArrayList Array mit variabler Größe
  schneller Zugriff über Indexfunktion
- ▷ Verwendet man die speziellen Funktionen der `LinkedList` nicht, kann man diejenige Listen-Klasse verwenden, die die bessere Performance bringt.

Iterator, Interface, das die Elemente einer Collection-Klasse nacheinander zur Verfügung stellt.

- □

Methoden von `Iterator`	Bedeutung
`boolean hasNext()`	`true`, wenn es noch mehr Elemente gibt
`Object next()`	Zeiger auf das nächste Element

- ● Jede `Collection` kann mit der Methode `iterator()` einen Iterator erzeugen, der ihre Elemente durchläuft.
- ▷ Mit Hilfe des Iterators kann man durch komplexe Strukturen wie Mengen oder Listen so einfach durchlaufen wie durch ein Array: mit einer `for`-Schleife.

□
```
import java.util.*;
class CollectionTest {
  public static void main(String[] args) {
    Collection c = new HashSet();
    Integer    o;
    Iterator   it;

    c.add(new Integer(3));
    c.add(new Integer(4));
    c.add(new Integer(6));
    c.add(new Integer(3));   // wird nicht eingefügt

    int sum = 0;
    for (it = c.iterator(); it.hasNext(); ) {
      o = (Integer) it.next(); // cast ist nötig
      System.out.println(o);
      sum += o.intValue();
    }
    System.out.println("Summe ist " + sum);
  }
}
```

Einfache Graphik

Alle modernen Betriebssysteme haben graphische Benutzer-Oberflächen , die auch zur Darstellung von Graphiken in Fenstern genutzt werden können. Java ermöglicht graphische Darstellungen in systemun-

abhängiger Weise auf allen Plattformen.
Frame, Klasse zur Darstellung eines Fensters mit Rahmen, Titelzeile und den üblichen Schaltflächen zum Verkleinern, Vergrößern und Schließen des Fensters.

□

Methoden von `Frame`	Bedeutung
`Frame(String name)`	legt Fenster-Objekt an noch keine Darstellung auf dem Bildschirm `name` erscheint in der Titelzeile
`setVisible(boolean b)`	Fenster sichtbar bzw. unsichtbar machen
`paint(Graphics g)`	zeichnet den Inhalt des Fensters wird automatisch aufgerufen beim (Wieder-)Aufbau des Fensters
`getSize()`	Größe des Fensters in Pixeln liefert Dimension-Objekt (Höhe `height`, Breite `width`)
`setSize(int w, int h)`	Größe des Fensters setzen kann auch dynamisch vom Benutzer geschehen

▷ Die `paint`-Methode liefert für `Frame` ein leeres Fenster mit Rahmen und Titelzeile. Man muß sie überschreiben, um eigene Graphiken zu erzeugen.

Graphics, Klasse, die die Zustandsinformationen und Methoden zum Zeichnen enthält.

□ Informationen in Graphics: Zeichenfarbe, Zeichenmethode (deckend, löschend-deckend), Font.
Methoden von Graphics: `drawLine`, `drawPolygon`, `drawOval`, `drawImage`, `drawString`, `fillPolygon`, `fillOval`

▷ Koordinaten werden als Integergrößen in Pixeln angegeben. Der Koordinatenursprung liegt zunächst in der linken, oberen Ecke, kann aber mit translate verschoben werden. Die x-Achse zeigt nach rechts, die y-Achse nach unten.

Event, Beschreibung eines Ereignisses, das an einem Objekt eingetreten ist. Insbesondere werden damit Aktionen gemeldet, die ein Benutzer einer graphischen Programmoberfläche an einer Komponente ausgelöst hat. Alle Events sind abgeleitet von der Klasse `EventObject`.

□

spezielles Event	Bedeutung
`WindowEvent`	Zustandsänderung eines Fensters (aktiviert, geschlossen)
`MouseEvent`	Mausklick oder Ziehen der Maus
`TextEvent`	Änderung des Inhalt eines Textfensters
`ActionEvent`	Eine Komponente wurde “betätigt” (Knopf gedrückt, Menüeintrag ausgewählt)

Listener, Objekt, das ein Event empfängt und daraufhin eine Aktion durchführt. Für jeden Eventtyp gibt es einen zugehörigen Listenertyp. Außerdem stehen Klassen zur Verfügung, die leere Methoden für alle benötigten Aktionen enthalten, die sogenannten **Adapterklassen**.

□ Die Klasse `WindowAdapter` enthält u.a. die (leeren) Methoden

```
windowOpened(WindowEvent e)
windowClosed(WindowEvent e)
windowIconified(WindowEvent e)
```

▷ Jedes Programm, das Fenster benutzt, braucht zumindest einen Listener, um auf das Schließen eines Fensters zu reagieren. Dazu leitet man eine Klasse von `WindowAdapter` ab und implementiert die Methode `windowClosing()`.

● Jede Komponente einer graphischen Oberfläche hat Methoden, um Listener anzumelden, die auf seine Events reagieren sollen.

□ Die Klasse `Frame` hat z.B. Methoden `addWindowListener` und `addMouseListener` für Fensteraktionen und Mausbewegungen.

☐ Das folgende Beispiel zeigt ein vollständiges Programm zur Anzeige des Graphen einer Funktion:

```
import java.awt.*;
import java.awt.event.*;

class WindowCloser extends WindowAdapter {

  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
}

public class DrawGraph extends Frame {

  public DrawGraph(String name) {
    super(name);
  }

  public void paint(Graphics g) {
    double s, t;
    int    ya, yb;

    // bestimme Größe der Zeichenfläche
    Dimension d  = getSize();
    int       dx = d.width;
    int       dy = d.height;

    // Skalierungsfaktoren von (t,s) zu (x,y)
    double  sx = 10.0/dx;    //  0 <= t <= 10
    double  sy = 2.0/dy;     // -1 <= s <=  1

    // Anfangswert
    ya = (int) Math.round((1.0 - myFunc(0))/sy);

    for (int x = 1; x <= dx; x++) {
      t = sx * x;
      s = myFunc(t);
      yb = (int) Math.round((1.0 - s)/sy);
      g.drawLine(x-1, ya, x, yb);
      ya = yb;
    }
  }

  double myFunc(double t) {
    return Math.cos(t)*Math.exp(-t/8.0) ;
  }

  public static void main(String[] args) {
    Frame frame = new DrawGraph("Graph");

    WindowListener l = new WindowCloser();
    frame.addWindowListener(l);

    frame.setSize(300, 300);
    frame.setVisible(true);
  }
}
```

- ▷ Das Beispielprogramm kann einfach erweitert werden, um die Darstellung von Graphen in beliebigen Intervallen zu erlauben.
- ▷ Nachteil des obigen Programms: Die zu zeichnende Funktion ist fester Bestandteil der `DrawGraph`-Klasse. Will man eine Klasse zur universellen Darstellung von Funktionen erzeugen, kann man dazu eine abstrakte Funktions-Klasse als Hilfsklasse für `DrawGraph` definieren, von der man eine Klasse mit der konkret zu zeichnenden Funktion ableitet.
- ▷ Eine graphische Benutzeroberfläche (**GUI**) ist typischerweise aufgebaut aus Fertigbausteinen wie Knöpfen, Auswahllisten oder Textfeldern. Java liefert eine große Sammlung von flexiblen Komponenten mit der Swing-Bibliothek (Paket `javax.swing`).

Component (Komponente), abstrakte Klasse zur Beschreibung von Objekten, die eine graphische Darstellung haben und Benutzereingaben verarbeiten.

□

GUI-Komponenten	Bedeutung
`JButton`	beschrifteter Knopf
`JTextField`	Feld zur Textein- und ausgabe
`JScrollBar`	Schiebebalken
`JComboBox`	Auswahlbox
`Canvas`	“Leinwand” = Bereich für eigene Graphik
`Container`	Komponente, die andere Komponenten enthält

- ▷ Die Methoden `paint`, `setSize`, `getSize` und `setVisible` der Klasse `Frame` stammen aus `Component` und stehen daher in allen Komponenten zur Verfügung.
- ● Die Container-Klassen ordnen ihre Unter-Komponenten nach bestimmten Regeln an. Dazu benutzen sie einen **LayoutManager**, der mit Hilfe der Methode `setLayout` festgelegt wird.
- □ Einfache Layout-Manager sind:

LayoutManager-Typ	Funktionsweise
`FlowLayout`	Komponenten beliebiger Größe hintereinander, bei Bedarf mehrere Zeilen
`GridLayout`	rechteckiges Gitter, dessen Zellen die Komponenten aufnehmen
`BorderLayout`	verteilt Komponenten auf ein Zentralgebiet und vier Ränder

□ In einem `JPanel`, der einfachsten Container-Klasse, soll ein Graphikbereich (`Canvas`) oben eine Überschrift und rechts einen Schaltknopf bekommen:

```
JPanel  p = new JPanel();
Canvas  c = new Canvas();
JButton b = new JButton("Drück mich");
JLabel  l = new JLabel("Dies ist kein Titel");

p.setLayout(new BorderLayout());
p.add("Center", c);
p.add("East", b);
p.add("North", l);
```

Applets

Applet, kleines Programm, das innerhalb eines Browser-Fensters abläuft.

- ▷ Ein Applet wird in eine HTML-Seite eingebettet und vom Browser beim Anzeigen dieser Seite automatisch ausgeführt.

□ Einbinden des Applets `GraphApplet` in eine HTML-Seite:

```
<HTML>
<HEAD>
<TITLE> Mein kleines Applet </TITLE>
</HEAD>
<BODY>
Das folgende Fenster zeigt mein kleines Applet: <BR>
<APPLET CODE="GraphApplet.class" WIDTH=400 HEIGHT=400>
</APPLET>
</BODY>
</HTML>
```

▷ Das Feld <APPLET> gibt an, welche Klasse geladen wird (`CODE="GraphApplet.class"`) und wie groß der Ausgabebereich des Applets innerhalb des Browser-Fensters sein soll (`WIDTH=400 HEIGHT=400`). Zum Aufbau einer HTML-Seite vgl. Abschnitt 24.4.5.

▷ Da Applets meistens direkt über das Internet geladen werden, müssen besondere Sicherheitsvorkehrungen getroffen werden, um bösartige oder versehentliche Beschädigungen des eigenen Systems zu verhindern.

● Einschränkungen für Applets:
- kein Zugriff auf lokale Dateien
- kein Start lokaler Programme
- kein Zugriff auf andere Rechner im Netzwerk
- Fenster außerhalb des Browser-Fensters nur mit Warnung

▷ Die Klasse `Applet` ist abgeleitet von `Panel`. Daher kann ein Applet wie alle Container-Klassen andere Komponenten einbinden (Methoden `add`, `setLayout`) und wie alle Componenten-Klassen auf Benutzer-Eingaben reagieren und zeichnen (Methoden `addMouseListener` etc., `repaint`).

□ Die folgenden `Applet`-Methoden werden bei Bedarf vom Browser aufgerufen:

Methoden von `Applet`	Bedeutung
`init`	beim Laden in den Browser
`start`	beim Anzeigen der HTML-Seite des Applets
`stop`	beim Anzeigen einer anderen Seite
`paint` (von `Container`)	zum Darstellen des Applets

▷ Alle diese Methoden sind in der Klasse `Applet` schon sinnvoll vordefiniert. Allerdings muß man mindestens eine der Methoden `init`, `start` oder `paint` überladen, wenn das Applet überhaupt etwas tun soll.

▷ Die Programmierung eines Applets unterscheidet sich folgendermaßen von der eines normalen Java-Programms:
- keine `main`-Methode
- Hauptklasse erbt von `Applet` statt von `Frame`
- kein `WindowListener` für `exit` nötig
- Setzen der Fenstergröße im HTML-File
- Zusammensetzen der Komponenten in `init` statt im Konstruktor

▷ In der Swing-Bibliothek gibt es eine an die übrigen Komponenten angepaßte Version des `Applet`, das `JApplet`. Im Unterschied zum normalen `Applet` verwaltet es seine Komponenten nicht direkt, sondern hat dafür ein sogenanntes **Content Pane**, eine übergeordnete Zeichenfläche, die man mit `getContentPane()` erhält.

□ Das folgende Applet zeichnet wie das obige Beispiel den Graphen einer gedämpften Schwingung. Hier kann man darüber hinaus mit einem Schaltknopf die Frequenz der Schwingung erhöhen.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class GraphApplet extends JApplet {

  private GraphCanvas   c;  // Zeichenfläche für den Graphen
  private ControlPanel  p;  // Panel für den Knopf

  public void init() {
    Container pane = getContentPane();
    pane.setLayout(new BorderLayout());

    // Zeichenfläche für den Graphen in der Mitte anordnen
    c = new GraphCanvas();
    pane.add("Center", c);

    // ControlPanel mit dem Button kommt nach unten
    p = new ControlPanel();
    pane.add("South", p);
  }

  class GraphCanvas extends Canvas {

    int frequency = 1;

    public void paint(Graphics g) {
      // s. vorheriges Beispielprogramm
    }

    double myFunc(double t) {
      return Math.cos(frequency*t)*Math.exp(-t/8.0) ;
    }

    public void increase() {
      frequency += 1;
      repaint();
    }
  }

  class ControlPanel extends JPanel {

    JButton   b;

    public ControlPanel() {
      b = new JButton("Schneller!");
      b.addActionListener(new MyButtonListener());
      add(b);
    }
  }

  class MyButtonListener implements ActionListener {

    public void actionPerformed(ActionEvent ev) {
      c.increase();
    }
  }
}
```

▷ Einige Bemerkungen zum Programm:

- Beim Drücken des `Button`s im `ControlPanel` wird ein `ActionEvent` ausgelöst. Daraufhin ruft der `ActionListener` die Methode `c.increase` auf, die die Frequenz der dargestellten Funktion erhöht.
- Die Klasse `MyButtonListener` hat als innere Klasse von `GraphApplet` direkten Zugriff auf den privaten `GraphCanvas c` und kann dessen Methode `increase()` aufrufen.
- Das `ControlPanel` benutzt als Standard-LayoutManager das `FlowLayout`, das dem Schaltknopf seine durch die Aufschrift gegebene Größe beläßt. Würde man den `Button` direkt in `GraphApplet` einfügen, bekäme er vom `BorderLayout` die Breite der Graphik.

Threads

Bei der Programmierung von graphischen Oberflächen muß man beachten, daß die Ausführung eines Kommandos nicht dazu führen darf, daß das Programm auf weitere Eingaben nicht mehr reagiert. Der Benutzer soll immer den Eindruck haben, daß jederzeit Eingaben möglich sind, die "sofort" Reaktionen auslösen. Zur Umsetzung dieser Idee der "gleichzeitigen" Ausführung verschiedener Programmteile (hier: GUI und ausgelöste Aktionen) dient das Konzept der Threads.

Thread, eine Folge von nacheinander ausgeführten Anweisungen innerhalb eines Programms. Ein Programm kann aus mehreren Threads bestehen, die aus der Sicht des Programmierers gleichzeitig ausgeführt werden.

▷ Bei einem Programm mit mehreren Threads wechselt die CPU normalerweise in schneller Folge zwischen den Threads hin und her, so daß der Eindruck von gleichzeitiger Abarbeitung entsteht. Auf einem Parallelrechner mit mehreren CPUs können Threads wirklich gleichzeitig ablaufen. Hier ist die Verwendung von Threads eine Methode, um die höhere Leistung des Parallelrechners mit einem Programm ausnutzen zu können.

▷ In Java ist `Thread` eine Klasse aus dem Standardpaket `java.lang`. Sie enthält u.a. die Methode `start`, die einen bereits initialisierten Thread zur Ausführung bringt und dessen `run`-Methode aufruft.

● Beim Start der virtuellen Java-Maschine wird automatisch ein Thread erzeugt, der die `main`-Methode der angegebenen Klasse ausführt.

● Ein Thread läuft solange, bis seine `run`-Methode beendet wird (direkt oder durch eine nicht abgefangene Ausnahme) oder bis `System.exit` aufgerufen wurde.

▷ Vorgehensweise zum Erzeugen von Threads:

Klasse erzeugen, die von `Thread` erbt und `run` überschreibt
innerhalb einer Methode (z.B. `main`):

Thread-Objekt erzeugen
Starten mit `thread.start`
Thread beginnt, führt `thread.run` aus
aufrufende Methode macht gleich weiter

□
```
class MyThread extends Thread {

  String  name;
  int     work;

  MyThread(String n, int w) {
    super();
    name = n;
    work = 1000*w;
  }

  public void run() {
    for (int i=0; i<3; i++) {
      System.out.println(name + ": " + i);
      // arbeite
      double t = 0.0;
      for (int j=0; j<work; j++) {
        t = Math.cos(t);
      }
    }
    System.out.println(name + ": ready");
  }
}

public class ThreadTest {
  public static void main(String[] args) {
    MyThread  t1 = new MyThread("Thread1", 10);
    MyThread  t2 = new MyThread("Thread2", 20);

    System.out.println("main: starting threads");
    t1.start();
    t2.start();
    System.out.println("main: ready");
  }
}
```

Dieses Programm kann bei jedem Lauf andere Ausgaben erzeugen, je nachdem, wie die Threads an die Reihe kamen, z.B:

```
main: starting threads
main: ready
Thread1: 0
Thread1: 1
Thread2: 0
Thread1: 2
Thread2: 1
Thread1: ready
Thread2: 2
Thread2: ready
```

▷ Es ist nicht festgelegt, nach welchem Verfahren Threads auf die CPU bzw. CPUs verteilt werden. Threads können aber über einige Methoden darauf Einfluß nehmen:

Methoden von `Thread`	Bedeutung
`static void sleep(long milli)`	Der aufrufende Thread wartet die angegebene Zeit (in Millisekunden), bevor er wieder ausgeführt wird.
`static void yield()`	Der aufrufende Thread pausiert kurz, um andere Threads an die Reihe kommen zu lassen.
`void join()`	Der aufrufende Thread wartet, bis der angegebene Thread (das `this` von `join`) beendet wurde.

▷ Dadurch, daß lauffähige Threads sich in unvorhersagbarer Weise bei der Ausführung abwechseln, können unerwartete Probleme auftauchen, wenn Threads gemeinsam an Objekten arbeiten.

□ Klasse `Konto` habe Methoden `hole` und `setze`, um den Kontenstand abzufragen und zu setzen. An zwei Bankautomaten, durch zwei Threads dargestellt, werden vom `Konto k` jeweils 100 Euro abgehoben:

```
int konto1 = k.hole();
konto1 -= 100;
k.setze(konto1);
```

Bei einem Kontostand von 1000 Euro kann dann folgendes passieren:

Thread1	Thread2	Bemerkung
`int konto1 = k.hole();`		konto1 = 1000
`konto1 -= 100;`		konto1 = 900
		Wechsel zu Thread2
	`int konto2 = k.hole();`	konto2 = 1000
	`konto2 -= 100;`	konto2 = 900
	`k.setze(konto2);`	Kontenstand: 900
		Wechsel zu Thread1
`k.setze(konto1);`		Kontenstand: 900 !!

▷ Damit sich Threads untereinander synchronisieren können, gibt es in der Sprache selbst und in der Basisklasse `Object` geeignete Vorkehrungen:

● Methoden können mit dem Attribut **synchronized** gekennzeichnet werden. Für ein Objekt kann höchstens ein Thread auf einmal eine solche Methode ausführen.

▷ Versuchen mehrere Threads, `synchronized`-Methoden eines Objekts auszuführen, bekommt einer den Zuschlag; alle anderen warten jeweils, bis der nächste fertig wird.

▷ Jedes Objekt hat ein spezielles Feld, den **Lock**, das den Zugang von Threads kontrolliert. Alle Synchronisierungs-Mechanismen beruhen darauf, daß immer nur ein Thread auf einmal Zugriff auf diesen Lock haben kann.

▷ Um zu verhindern, daß die Threads ständig warten, muß der synchronisierte Bereich möglichst klein sein. Dazu kann man statt einer ganzen Methode auch nur einige Anweisungen vor gleichzeitigem Zugriff schützen.

Synchronisations-Anweisung:

```
synchronize(Ausdruck)
  Anweisung;
```

Ausdruck muß ein gültiger Zeiger auf ein Objekt sein. Ein Thread versucht, den Lock dieses Objekts zu bekommen, und führt dann die Anweisung aus.

▷ Ein weiteres Synchronisationsverfahren besteht darin, daß ein Thread auf das Eintreten einer Bedingung wartet, bis ein anderer Thread signalisiert, daß diese eingetreten ist.

- Folgende Methoden zur Synchronisation von Threads sind in der Basisklasse `Object` definiert:

Methoden von `Object` zur Synchronisation	Bedeutung
`void wait()`	Der aufrufende Thread wartet, bis ein anderer Thread die `notify`- oder `notifyAll`-Methode des benutzten Objekts aufruft.
`void notify()`	Weckt einen der Threads auf, die die `wait`-Methode dieses Objekts aufgerufen haben.
`void notifyAll()`	Weckt alle Threads auf, die die `wait`-Methode dieses Objekts aufgerufen haben.

  Alle drei Methoden dürfen nur innerhalb von synchronisierten Bereichen (bzw. synchronisierten Methoden) aufgerufen werden. Mit dem Aufruf von `wait()` wird der Lock freigegeben. Ein geweckter Thread versucht als erstes, den Lock wiederzuerlangen.

  - ▷ Ein geweckter Thread kann insbesondere erst dann weitermachen, wenn der Thread, der ihn geweckt hat, den Lock wieder freigibt, d.h. den synchronisierten Bereich verläßt.

  - □ Ein wichtiger Anwendungsfall für die Synchronisation mit `wait`/`notify` ist das Erzeuger-/Verbraucher-Problem: Ein Thread (der Erzeuger) schreibt Zwischenergebnisse in einen Pufferspeicher, ein zweiter (der Verbraucher) liest sie daraus zur weiteren Verarbeitung. Der Erzeuger muß warten, wenn der Pufferspeicher voll ist, der Verbraucher, wenn er leer ist.

  - ▷ Benötigen Threads mehrere Locks auf einmal, kann folgende Schwierigkeit auftreten:
    Thread 1 hat Lock1 und wartet auf Lock2, Thread2 hat Lock2 und wartet auf Lock1.
    Eine solche Situation nennt man **Verklemmung** (Deadlock). Sie zu vermeiden gehört zu den schwierigsten Problemen beim Programmieren mit Threads.

Einführung in Fortran

Fortran wurde schon in den fünfziger Jahren als eine der ersten Programmiersprachen zur Lösung naturwissenschaftlicher und technischer Probleme konzipiert. Sie wurde immer wieder an aktuelle Entwicklungen angepaßt; entsprechende Versionen wurden als internationale (ISO) und entsprechende nationale Normen (ANSI, DIN) festgelegt. Die zur Zeit aktuelle Version heißt Fortran 95 und ist Grundlage der folgenden Beschreibung.

▷ Da sich beim Übergang von FORTRAN77 zu Fortran 90 große Änderungen ergeben haben, wird i.f. gelegentlich auf wichtige Unterschiede zu FORTRAN77 explizit hingewiesen.

a) Programmaufbau

- Fortran-Anweisungen stehen in Zeilen, die bis zu 132 Zeichen lang sein dürfen. Sie können verlängert werden, indem als letztes Zeichen ein & angefügt wird. Die nachfolgende Zeile ist dann eine Fortsetzungszeile.
- Kommentare werden mit `!` eingeleitet. Sie erstrecken sich immer bis zum Ende der Zeile.
- Namen zur Bezeichnung etwa von Variablen oder Routinen bestehen aus 1 bis 31 Zeichen, wobei Buchstaben, Zahlen und das Symbol _ verwendet werden können. Das erste Zeichen muß ein Buchstabe sein. Es wird nicht zwischen Groß- und Kleinbuchstaben unterschieden.

▷ Ein Programm enthält genau ein Hauptprogramm, das mit einer Zeile

```
program PROGRAMMNAME
```

beginnen sollte. Danach können Unterprogramme (Subroutines, Functions) folgen, die auch in eigenen Dateien stehen dürfen.

- Ein Haupt- oder Unterprogramm beginnt mit Spezifikations-Anweisungen (z.B. Beschreibung eigener Datentypen, Deklaration von Variablen), denen ausführbare Anweisungen folgen. Es endet mit `end`.

▷ Optional kann man an jedes `end` noch anfügen, was es beendet, z.B.

```
end program test
end subroutine init
```

Dies erhöht, vor allem bei stärkeren Verschachtelungen, die Lesbarkeit des Programms.

b) Datenstrukturen

Einfache Datentypen, Darstellung von ganzen, reellen oder komplexen Zahlen, Zeichen oder logischen Werten.

- Fortran stellt die folgenden einfachen Datentypen zur Verfügung:
  `integer`, `real`, `complex`, `character`, `logical`

▷ Der Speicherbedarf und damit auch der Darstellungsbereich einer Größe eines solchen Typs sind maschinenabhängig. Für den Typ `real` gibt es mindestens zwei Varianten verschiedener Genauigkeit, meistens auch für die anderen Typen. Die benötigte Genauigkeit wählt man dann durch Angabe des `kind`-Parameters:

```
real(kind=4), real(kind=8), integer(kind=2)
```

Die Werte des `kind`-Parameters und ihre genaue Bedeutung sind vom Compiler abhängig.

▷ Mit Hilfe von vordefinierten (intrinsischen) Funktionen kann auf portable Weise eine bestimmte Genauigkeit ausgewählt werden, z.B. erhält man mit

```
REAL8 = selected_real_kind(12)
real(kind=REAL8) high_precision
```

eine Variable mit mindestens 12 Dezimalstellen, analog zum Typ `DOUBLE PRECISION` in FORTRAN77. Konstante wie `REAL8` sollte man in einem Modul sammeln und konsequent einsetzen.

□ Konstanten der jeweiligen Datentypen:

Datentyp	Beispiele
`integer`	`42, 0, -333`
`integer(kind=2)`	`17_2`
`real`	`3.141, -.1999, 2.7e9`
`real(kind=REAL8)`	`3.1415926_REAL8`
`complex`	`(2.0, 3.1e1), (0.0, 1.0)`
`character`	`'a', 'B', '$'`
`logical`	`.false., .TRUE.`

Variablendeklaration, Zuweisung von Variablen zu einem Typ. Konstante werden durch das zusätzliche Attribut `parameter` gekennzeichnet.

□
```
integer                      ::  i, j, zwei
real(kind=REAL8)             ::  a, b
character                    ::  in, out
real(kind=REAL8), parameter  ::  pi = 3.1415926
```

▷ Um die Kompatibilität zu älteren Fortran-Standards zu wahren, werden undeklarierten Variablen aufgrund ihres Anfangsbuchstabens Typen zugewiesen. Man schaltet diesen fehlerträchtigen Mechanismus aus und erzwingt eine explizite Deklaration aller Variablen, indem man Haupt- und Unterprogramme beginnt mit

```
implicit none
```

▷ Werden Variablen eines der Typen `integer`, `real` oder `complex` Werte eines anderen solchen Typs zugewiesen, erfolgt eine automatische Typumwandlung, die u.U. mit Verlust von Genauigkeit (Abschneiden von Stellen) verbunden ist.

▷ Vermischen von Datentypen ist häufige Fehlerursache, etwa:

```
integer  :: n
real     :: x

n = 42
x = n/100   ! x=0
```

▷ Man sollte *beabsichtigte* Typumwandlungen in seinem Programm dokumentieren, indem man eine der expliziten Funktionen zur Umwandlung benutzt, z.B. im obigen Beispiel:

```
x = real(n)/100.0
```

Fortran unterstützt die wichtigsten zusammengesetzten Datentypen, nämlich

- Felder
- Zeichenketten
- Strukturen
- Zeiger

Feld (**Array**), Datentyp, der mehrere Elemente des gleichen Typs in ein- oder mehrdimensionaler Anordnung zusammenfaßt.

- Der Zugriff auf die einzelnen Elemente erfolgt durch Index-Variable vom Typ `integer`.

□ Definition von Arrays:

```
real,    dimension(100)   :: v
integer, dimension(-5:5)  :: p
real,    dimension(10,10) :: matrix
```

Zugriff auf diese Arrays:

```
v(1)        = p(-5)
matrix(7,9) = v(100)
```

▷ Wird bei der Definition kein unterer Indexwert angegeben, fängt der Indexbereich bei 1 an.

▷ Der Zugriff auf ein Array mit einem Index außerhalb des gültigen Bereichs ist einer der häufigsten Fehler. Oft haben Compiler Optionen, um solche Fehler beim Programmablauf festzustellen.

▷ Zweidimensionale Arrays werden als Folge von Spalten abgespeichert, d.h. nach `a(1,1)` kommt `a(2,1)`. Dies kann zu Problemen führen bei der Verwendung von Bibliotheks-Routinen in anderen Programmiersprachen, die Arrays als Folge von Zeilen im Speicher erwarten (wie z.B. in C).

□ Arrays können auf verschiedene Weise initialisiert werden:

```
integer, dimension(5)    :: a = (/ 1, 2, 3, 4, 5 /)
real, dimension(100)     :: b = (/ (0.1*i, i=1,100) /)
real, dimension(100,100) :: c = 0.0
```

▷ Ein Array kann als ganzes Objekt auftreten, z.B. bei Zuweisungen oder arithmetischen Operationen:

```
a = b + c     ! Addition kompletter Arrays
```

- Man kann aus Arrays Teile herausgreifen, indem man jeweils den Anfangs- und Endindex und die Sprungweite in der Form `a(anfang:ende:sprungweite)` angibt.

▷ Für einen fehlenden Anfangs- bzw. Endindex wird der kleinste bzw. größte Wert gesetzt. Wird die Sprungweite nicht angegeben, ist sie 1.

```
real, dimension(-20:20) :: v
real, dimension(3, 4) :: a
v(2:10:3) == (/ v(2), v(5), v(8) /)
v(:12:10) == (/ v(-20), v(-10), v(0), v(10) /)
a(:, 3)   == (/ a(1,3), a(2,3), a(3,3) /)
```

▷ Möchte man Operationen nur mit bestimmten Elementen eines Arrays ausführen, kann man die `where`-Anweisung verwenden.

□
```
real, dimension(100,100) :: a
where (a > 0.5)
  a = a - 0.5
elsewhere
  a = 2*a
end where
```

- Kennt man die Größe eines Arrays erst zur Laufzeit des Programms, kann man es als `allocatable` vereinbaren und ihm später den benötigten Speicherplatz zuweisen.

□
```
real, allocatable, dimension(:,:) :: a
integer n, m

read *, n, m
allocate(a(n,m))
```

▷ Neben modernen Datentypen sind dynamische Arrays eine der wichtigsten Neuerungen gegenüber FORTRAN 77. Man sollte sie konsequent einsetzen, um Arraygrößen an die Problemgröße anzupassen, anstatt "auf Vorrat" riesige Matrizen anzulegen.

Zeichenkette (**String**), `character`-Variable mit einem zusätzlichen ganzzahligen Parameter, der die Anzahl der Zeichen angibt.

□
```
character(len=80) :: zeile
character(len=10) :: wort = 'juhu'   ! der Rest enthält Leerzeichen
```

● Ähnlich wie bei Arrays können aus einem String Teilzeichenketten gebildet werden.

□
```
character(len=16) :: zitat = 'Habe nun ach ...'
zitat(4:8)   ! 'e nun'
zitat(:4)    ! 'Habe'
```

▷ Strings sind *keine* Arrays. Insbesondere kann man nicht wie bei einem Array auf ein einzelnes Zeichen zugreifen:

```
zitat(4)     ! verboten
zitat(4:4)   ! = 'e'
```

Struktur (**Record**), Datentyp zur Zusammenfassung von Elementen verschiedener Typen zu einem Objekt.

□ Definition des Datentyps `Person`:

```
type Person
  character(len=20)  :: name
  integer            :: age
  real               :: height
end type Person
```

Definition von Variablen des Typs `Person`:

```
type(Person)  :: peter
type(Person)  :: mr_president = Person('Bill', 53, 1.78)
```

Zugriff auf die Komponenten:

```
peter%age = peter%age + 1
```

Zeiger (**Pointer**), Größen, die auf andere Variable verweisen. Sie haben neben dem Datentyp des Objekts, auf das sie zeigen, das Attribut `pointer`.

□ Definition von Pointern:

```
integer, pointer :: int_pointer
real(kind=long), dimension(:,:), pointer ::  pa, pb
```

● Ziele von Pointern müssen mit dem `target`-Attribut gekennzeichnet werden:

```
real, dimension(10, 10), target :: a
```

▷ Pointern können Ziele oder andere Pointer zugewiesen werden, oder sie bekommen dynamisch Speicher.

□
```
pa => a
pa(5,5) = 314.59          ! ändert den Wert von a(5,5)

allocate(pb(5, 7))
pb(2,3) = pa(3,2)
deallocate(pb)
```

▷ Programmierfehler bei der Verwaltung von dynamischem Speicher führen oft zu seltsamen Effekten und sind schwer aufzuspüren.

□ Die Kombination von Strukturen und Zeigern erlaubt die Definition von dynamischen Datenstrukturen wie verketteten Listen oder Baumstrukturen.

```
type List_Entry
   integer                   :: value
   type(List_Entry), pointer :: next
end type List_Entry

type(List_Entry), pointer   :: first, current

! Anfang und ein weiteres Element vorschalten
allocate(first)
first = List_Entry(17, null())       ! null() erst in Fortran 95
allocate(current)
current = List_Entry(19, first)      ! current%next => first
first   => current
```

c) Operatoren

□ Die folgende Tabelle enthält die wichtigsten Operatoren, nach ihrer Priorität fallend geordnet. Operatoren innerhalb der einzelnen Abschnitte besitzen gleiche Priorität.

Symbole	Bedeutung
**	Potenzierung
+, -	positives/negatives Vorzeichen
+, -	Addition, Subtraktion
//	Verkettung von Strings
==, / =	Gleichheit, Ungleichheit
>, >=, <, <=	Vergleichsoperatoren
.and.	logisches UND
.or.	logisches ODER

▷ Operatoren können auch Arrays von gleicher Form verbinden, sie gelten dann für alle Elemente einzeln.

▷ Man kann die Definition der vorgegebenen Operatoren auch auf eigene Datentypen ausdehnen oder sogar eigene Operatoren definieren. Dies geschieht typischerweise in Modulen.

d) Kontrollstrukturen

Bedingte Anweisung, knüpft die Ausführung von Anweisungen an Bedingungen:

```
if (logischer Ausdruck) then
  Block 1
else
  Block 2
end if
```

Die Kommandos von `Block1` werden nur ausgeführt, wenn der logische Ausdruck den Wert `.true.` hat, sonst wird `Block2` ausgeführt.

- ▷ Der `else`-Zweig kann entfallen.

Verzweigung, ermöglicht die Auswahl unter einer Reihe von Optionen:

```
select case (Auswahl-Ausdruck)
  case (Auswahl 1)
     Block 1
  ...
  case (Auswahl n)
     Block n
  case default
     Block n+1
end select
```

Der Auswahl-Ausdruck kann vom Typ `integer` oder `character` sein. Die einzelnen Auswahlen sind komma-getrennte Listen von Werten, wobei wie bei Array-Indizes auch Intervalle in der Form `wert1:wert2` möglich sind. Die einzelnen Auswahlen dürfen sich nicht überlappen. Der Zweig `default` erfaßt alle nicht explizit abgefragten Fälle.

- □
```
select case (ganze_zahl)
  case (:-1)
    vorzeichen = -1
  case (0)
    vorzeichen = 0
  case (1:)
    vorzeichen = 1
end select
```

Schleife, wiederholt eine Folge von Anweisungen solange, bis eine Abbruchbedingung erfüllt ist:

```
do
  Anweisungen1
  if (Abbruch-Bedingung) then
    exit
  end if
  Anweisungen2
end do
```

- ▷ Die Sonderfälle der **abweisenden** bzw. **nichtabweisenden Schleife** erhält man, wenn man `Anweisungen1` bzw. `Anweisungen2` wegläßt.
- □ Suchen in einer verketteten Liste:

```
current => first
do
  if (current%value = 42) then
    exit
  end if
  current => current%next
end do
```

▷ Dieses Beispiel versagt, wenn der Wert `42` in der Liste nicht vorkommt!

Laufanweisung, eine besondere Form der Schleife, bei der eine Indexvariable von einem Anfangs- zu einem Endwert herauf- oder heruntergezählt wird:

```
do index = start, end, schritt
  block
end do
```

Der Block wird wiederholt durchlaufen, wobei die `integer`-Variable `index` zunächst den Wert `start` hat und für jeden Durchlauf um `schritt` erhöht wird. Die Schleife wird beendet, wenn `index` den Wert `end` überschreitet. Ist `start > end`, wird der Block gar nicht durchlaufen.

▷ Bei `schritt < 0` gilt der letzte Absatz sinngemäß für die Rückwärtsschleife.

▷ Läßt man `schritt` weg, wird mit der Schrittweite 1 gerechnet.

□ Summe der ersten n ungeraden Zahlen:

```
sum = 0
do i=1, n
  sum = sum + (2*i - 1)
end do
```

▷ Einfache Schleifen lassen sich oft kürzer und übersichtlicher durch Array-Ausdrücke ersetzen.

e) Programmeinheiten

● Ein Fortran-Programm besteht aus mehreren **Programmeinheiten**, darunter genau einem Hauptprogramm, Unterprogrammen und Modulen.

▷ Die verschiedenen Programmeinheiten können sich auf eine oder mehrere Dateien verteilen.

● Die erste Zeile einer Programmeinheit beschreibt ihren Typ und gibt ihr einen Namen. Alle Programmeinheiten enden mit der `end`-Anweisung, die noch den Typ und den Namen der Einheit enthalten kann.

□
```
program intervall
 ...
end program intervall

subroutine init
 ...
end subroutine init

module lists
 ...
end module lists
```

● Die Ausführung des gesamten Programms endet, wenn eine `stop`-Anweisung ausgeführt wird. Die Ausführung eines Unterprogramms bzw. einer Funktion endet mit dem Ausführen der `return`-Anweisung, die Kontrolle geht dann an die aufrufende Programmeinheit zurück.

▷ Fortran unterscheidet bei Unterprogrammen zwischen `subroutine` und `function`. Eine `subroutine` gibt keinen Wert zurück und wird aufgerufen mit

```
call sub-name(parameterliste)
```

Eine `function` gibt direkt einen Wert zurück wie in

```
a = func-name(parameterliste)
```

- Bei der Definition einer Funktion kann der Funktionsname wie eine Variable benutzt werden, die den zurückzugebenden Funktionswert enthält. Man kann aber auch einen anderen Namen für den Rückgabewert wählen. Dies ist insbesondere bei rekursiven Funktionen nötig.

□
```
recursive function factorial(n) result(res)
  implicit none
  integer :: n, res

  if (n == 1) then
    res = 1
  else
    res = n*factorial(n-1)
  end if
end function factorial
```

- Mit dem **intent**-Attribut kann man angeben, ob ein Parameter nur zur Eingabe bzw. nur zur Ausgabe oder zu beidem verwendet wird.

□
```
integer function foo(a, b, c)
  implicit none
  real, intent(in)    :: a    ! a darf nicht geaendert werden
  real, intent(inout) :: b    ! b wird gelesen und geschrieben
  real, intent(out)   :: c    ! c wird nicht gelesen
```

▷ In Unterprogrammen, die Matrizen als Parameter haben, kann man die Dimensionen bei der Deklaration durch einen Doppelpunkt ersetzen, sie brauchen nicht explizit übergeben zu werden. Solche Arrays heißen **assumed shape arrays**.

▷ Mit Hilfe der `size`-Routine kann man die Array-Dimensionen im Unterprogramm erhalten, falls man sie explizit benötigt.

□
```
real, dimension(100, 100)  :: y1
real, dimension(5, 5)      :: y2
call juhu(y1)
call juhu(y2)
...

subroutine juhu(a)
  real, dimension(:,:)  :: a
  integer n, m

  n = size(a, 1)
  m = size(a, 2)
```

▷ Verwendet man beim Aufruf einer Routine für einen Parameter einen anderen Typ als bei ihrer Deklaration, kann es zu schwer zu findenden Fehlern bei der Programmausführung kommen. Schon der Compiler kann solche Fehler finden, wenn man im aufrufenden Programm die Parameterdeklarationen des Unterprogramms in einem `interface`-Block angibt. In einigen Fällen (z.B. bei assumed shape arrays) sind `interface`-Blöcke obligatorisch.

□ Ein Interface-Block wiederholt einfach die Deklarationen der Parameterliste:

```
interface
  integer function foo(a, b, c)
    real, intent(in), dimension(:,:)  :: a
    real, intent(inout)               :: b
    real, intent(out)                 :: c
  end function foo
end interface
```

▷ Interface-Blöcke verdoppeln geschriebenen Code und können dadurch neue Fehler erzeugen. Man kann diese Arbeit dem Compiler überlassen, indem man alle Unterprogramme in ein oder mehrere Module verpackt.

▷ Ein Unterprogramm mit “assumed shape arrays” benötigt häufig lokale Matrizen, deren Dimensionen von der Größe der Eingangsmatrizen abhängen (“workspace”). Solche Matrizen werden als **automatische Arrays** bezeichnet. Sie werden folgendermaßen deklariert:

```
subroutine juhu(a)
  real, dimension(:)          :: a
  real, dimension(2*size(a))  :: w      ! workspace
```

Modul, Programmeinheit zur Zusammenfassung von Typ- und Unterprogramm-Definitionen sowie globaler Daten:

```
module meinmodul
  ...
end module meinmodule
```

● Mit dem Kommando

```
use module-name
```

können alle Moduldaten einer anderen Programmeinheit zugänglich gemacht werden.

▷ Module, die nur globale Daten enthalten, ersetzen die `COMMON`-Blöcke älterer Fortran-Versionen, und bieten zusätzlich garantierte Typsicherheit.

□ Häufig wird ein Modul eingesetzt, um `KIND`-Parameter zu definieren, die einfache Datentypen mit vorgegebener Genauigkeit bezeichnen:

```
module kinds
  integer, parameter :: INT4  = selected_int_kind(9)
  integer, parameter :: REAL4 = selected_real_kind(5)
  integer, parameter :: REAL8 = selected_real_kind(12)
end module kinds

   ...
  use kinds
  real(kind=REAL8)  :: a, b
```

▷ Mit Hilfe von Modulen kann man eigene Datentypen zusammen mit den Operationen auf ihnen zusammenfassen. Routinen, die nur innerhalb des Moduls verwendet werden sollen, können durch das `private`-Attribut nach außen unsichtbar gemacht werden.

□
```
module bruchrechnung
  type bruch
    integer  :: zaehler, nenner
  end type bruch

  interface operator(*)
    module procedure ratmul
  end interface

  private kuerzen

 contains

  type(bruch) function ratmul(r1, r2)  ! Produkt zweier Brüche
    implicit none
    type(bruch), intent(in) ::  r1, r2
    ratmul%zaehler = r1%zaehler * r2%zaehler
    ratmul%nenner  = r1%nenner  * r2%nenner
    call kuerzen(ratmul)
  end function ratmul

  subroutine kuerzen(r)
       ...
  end subroutine kuerzen
end module bruchrechnung

program bruchtest
  use  bruchrechnung
  type(bruch)   :: a,b,c
  c = a * b
end program bruchtest
```

f) Ein- und Ausgabe

● Ein- und Ausgabe von Daten geschieht mit Hilfe der `read`- bzw. `write`-Anweisung, die folgende Form haben:

```
read  (unit, format) variable_1, ... variable_n
write (unit, format) variable_1, ... variable_n
```

`unit` ist ein Integerwert, der auf eine - vorher geöffnete - Datei verweist. `format` ist eine Zeichenkette, die die Formatierung der Daten beschreibt.

▷ Ersetzt man `unit` durch `*`, wird die Standard-Eingabe- bzw. Standard-Ausgabe-Einheit verwendet - in der Regel ist dies das Terminal.

▷ Ersetzt man `format` durch `*`, werden systemabhängige Standardwerte benutzt (**listengesteuerte Ein-/Ausgabe**). Bei der Eingabe werden dabei die gängigen Schreibweisen, z.B. von `real`-Werten, erkannt.

▷ Für den häufigen Fall der listengesteuerten Ein-/Ausgabe auf die Standard-Einheiten verwendet man besondere Kurzformen:

```
print *, a, i, 'juhu'
read *, z1, z2
```

□ Zur genauen Kontrolle des Formats gibt es eine Fülle von Formatbeschreibern, mit denen man Form und Länge der einzelnen Ein-/Ausgabefelder genau festlegen kann. Die wichtigsten zeigt das folgende

Beispiel:

```
a = 'juhu'
i = 42
f = 3.1415926
write (*, '(A6, I5, F10.5, A1, E10.4)') a, i, f, '/', f
```

Ausgabe:

```
  juhu   42   3.14159/0.3142E+01
```

▷ Zur einfachen Ausgabe von Arrays kann man **implizite do-Schleifen** verwenden:

```
print *, (( a(i,j), i=1,10), j=1,10)
```

● Will man in eine Datei schreiben oder aus ihr lesen, muß man sie vorher mit der `open`-Anweisung öffnen. Wird sie nicht mehr gebraucht, wird sie mit der `close`-Anweisung geschlossen.

▷ Bei der Bearbeitung von Dateien muß man sich Gedanken über mögliche Fehler machen, etwa falsche Eingabedaten oder Probleme beim Schreiben. Dazu kann man der `read`- bzw. `write`-Anweisung den `iostat`-Parameter mitgeben, der positiv wird, wenn ein Fehler aufgetreten ist, negativ, wenn das Ende des Files erreicht wurde, und 0, wenn alles in Ordnung war.

□
```
open(1, 'array.dat')
do i=1, 100
   read(1, *, iostat=status) a(i)
   if (status /= 0) exit
end do
if (status > 0) write (*,*) 'Lesefehler!'
close(1)
```

g) Standard-Routinen

▷ Fortran enthält eine große Anzahl von Standard-Routinen, vor allem für die wichtigsten mathematischen Funktionen, für Array- und String-Operationen und zum Abfragen von Objekt-Eigenschaften. Die folgende Tabelle stellt einige wichtige Vertreter dieser Klassen vor.

□

Routine	Bedeutung
`exp`, `log`, `sqrt`	Exponentialfunktion, Logarithmus, Wurzel
`sin`, `cos`, `tan`, `asin`, `sinh`, ...	trigonometrische/hyperbolische Funktionen und Inverse
`min`, `max`, `abs`	Minimum, Maximum, Absolutbetrag
`floor`, `ceiling`	nächst kleinere/größere ganze Zahl
`random_number`	Zufallszahl zwischen 0.0 und 1.0
`dot_product`	Skalarprodukt von Vektoren
`matmul`	Matrixmultiplikation
`transpose`	Transponierte einer Matrix
`sum`, `product`, `maxval`	Summe/Produkt/Maximum aller Matrixelemente
`lgt`	lexikalischer Vergleich von Strings
`index`	Teilstring in einem String finden
`trim`	führende Leerzeichen entfernen
`selected_real_kind`	KIND-Parameter mit vorgegebener Genauigkeit
`size`	Dimensionen eines Arrays
`allocated`	Zustand eines allozierbaren Arrays

▷ Die mathematischen Funktionen können auch mit Array-Argumenten benutzt werden. Sie werden dann auf jedes Array-Element einzeln angewandt.

24.4 Informationen im Internet

Das World Wide Web

Die Möglichkeiten, die sich aus der Verbindung von Rechnern ergeben, wurden zuerst im militärischen und wissenschaftlichen Umfeld eingesetzt. Erst mit der Entwicklung von Programmen, die einen einfachen und unmittelbaren Zugang zu einer Vielfalt miteinander verknüpfter Informationen erlaubte, wurde das Netzwerk in seiner Form des "World Wide Web" zu einem Massenmedium.
Voraussetzung für große Computer-Netzwerke ist neben der technischen Infrastruktur aus Leitungen und Vermittlungsrechnern eine allgemein verbreitete Sprache zur Verbindungsaufnahme und zum Austausch von Daten, ein **Protokoll**. Weltweit durchgesetzt hat sich inzwischen ein spezielles Protokoll namens **TCP/IP**.

Internet, Gesamtheit der weltweit miteinander verbundenen Rechner, die über das TCP/IP-Protokoll Daten austauschen.

- Wichtiger Bestandteil des TCP/IP-Protokolls ist die Kennzeichnung von Empfänger und Absender durch weltweit eindeutige Nummern, die **IP-Adressen**. Zur Vereinfachung für den Benutzer werden diesen Nummern auch Namen (sogenannte **Domain-Namen**) zugeordnet, die folgendermaßen aufgebaut sind:

  `RECHNERNAME.UNTERNETZ.NETZ.HAUPTNETZ`

  Dabei bezeichnet `RECHNERNAME` den eigentlichen Rechner, `UNTERNETZ` eine Untergliederung von `NETZ`, `HAUPTNETZ` eine globale Einteilung aller angeschlossenen Teilnetze. Je nach Größe (Anzahl der Rechner) von `NETZ` kann das `UNTERNETZ` auch entfallen bzw. noch weiter untergliedert werden.

- Die Adresse

  `pc42.th.physik.uni-frankfurt.de`

  bedeutet:

pc42	Rechnername
th	Unterunternetz
physik	Unternetz
uni-frankfurt	Netz
de	Hauptnetz

  - Die Hauptnetze (**Top-Level-Domains**) unterscheiden zwischen Netzen in verschiedenen Ländern und (historisch bedingt) Organisationen in den USA. An der Top-Level-Domain kann man häufig den groben geographischen Standort eines Rechners ablesen.

Top-Level-Domain	Bedeutung
de	Deutschland
fr	Frankreich
uk	Großbritannien
jp	Japan
edu	Hochschule/Forschungseinrichtung in den USA
gov	US-Regierung
com	Firmen (auch weltweit)

  - Die Rechnerkommunikation auf der Ebene des TCP/IP-Protokolls bleibt für den Anwender in der Regel unsichtbar. Er ist an der Nutzung spezieller Netzwerk-Anwendungen (**Internet-Dienste**) interessiert.

- Wichtige Internet-Dienste sind:

  E-Mail Verschicken von Nachrichten zwischen Personen

News lokal oder weltweit verbreitete schwarze Bretter zur - meistens öffentlichen - Kommunikation

FTP Übertragung von Dateien (Daten, Programmen) zwischen Rechnern

Telnet Anmelden und Arbeiten auf einem entfernten Rechner

- ▷ Mit der Zunahme der im Internet auf vielen Rechnern verfügbaren Informationen wurde es zunehmend schwierig, gezielt bestimmte Daten zu finden. Daher wurde am europäischen Kernforschungszentrum CERN ein bahnbrechendes System entwickelt, das die Integration verschiedenster Dokumente mit beliebigen Querverweisen gestattete: das **World Wide Web** (**WWW**).
- ● Kernelemente des WWW sind:
  1. ein Protokoll zur Übertragung verschiedenartigster Daten von einem Server (dem **WWW-Server**) zum Rechner des Anwenders (das **Hypertext Transfer Protocol**, kurz **http**),
  2. eine Sprache zur Beschreibung des Inhalts eines Dokuments einschließlich vorhandener Querbezüge zu anderen Dokumenten (**HTML**),
  3. ein Programm zum Anzeigen von HTML-Dokumenten und zum einfachen Verfolgen von Querverweisen, auch zu Dokumenten auf völlig anderen Rechnern (der **Browser**).
- ▷ Sowohl die Protokolle und Methoden als auch die notwendigen konkreten Programme wurden vom CERN und anderen wissenschaftlichen Instituten entwickelt und frei der Allgemeinheit zur Verfügung gestellt. Sie fanden schnell weltweite Verbreitung und lösten damit den gegenwärtigen Internet-Boom aus.
- ▷ Mit der rasanten Entwicklung des WWW entstanden auch völlig neue Anwendungsfelder. Insbesondere wird es zunehmend zu kommerziellen Zwecken wie dem Vertrieb von Software und Dienstleistungen aller Art inklusiver finanzieller Transaktionen genutzt. Dies erfordert die Entwicklung neuer Methoden wie Verschlüsselungs- und Abrechnungsmechanismen - was von der ursprünglichen, auf freie Informationsausbreitung bedachten Internetgemeinde mißtrauisch beäugt wird.

Browser

- ▷ Der erste graphische Browser war das Programm Mosaic, das am amerikanischen Supercomputer-Zentrum NCSA entwickelt worden war. Aufgrund seiner leichten Bedienbarkeit verbreitete er sich schnell und ermöglichte damit den Durchbruch des WWW zur Massenanwendung.
- ▷ Die derzeit am weitesten verbreiteten Browser sind der Netscape Navigator der Firma Netscape und der Internet Explorer von Microsoft. Während der Internet Explorer hauptsächlich auf Windows-Plattformen verbreitet ist und erst seit kurzem auch für andere Betriebssysteme portiert wird, ist der Netscape Navigator auf allen verbreiteten Systemen verfügbar.
- ▷ In Folge des Konkurrenzkampfes wurden beide Programme mit immer mehr Funktionen ausgestattet und sind heute Universal-Programme für Netzanwendungen geworden.
- □ Einige Zusatzfunktionen von Netscape bzw. Internet Explorer: Verwaltung von Email und News, graphischer Editor zum Erstellen von HTML-Dokumenten, netzwerkfähiger Terminkalendar, Online-Videokonferenzsystem.
- ▷ Beide Browser führten jeweils verschiedene Erweiterungen des HTML-Standards ein, die ein genaueres Layout oder dynamisch sich ändernde HTML-Seiten ermöglichen sollen. Damit sind HTML-Seiten entstanden, die sich nur noch mit dem Browser einer Firma ansehen lassen - ganz im Gegensatz zur Idee des WWW.
- ▷ Der HTML-Standard 4.0 integriert viele der neuen Funktionen in hersteller-übergreifender Weise. Allerdings wird er noch nicht vollständig von den Browsern unterstützt.
- ▷ Ein Sonderfall ist der frei erhältliche Browser **Lynx**, der ohne graphische Oberfläche auf Textbasis arbeitet. Er ist damit für einfache Terminals oder - etwa mit einer Braille-Zeile oder einer Vorlesefunktion - für Blinde geeignet. Die zunehmende Multimedia-Orientierung im WWW schränkt seinen

Nutzen aber sehr ein.

Nutzung der Internet-Dienste

a) WWW

URL (**Uniform Resource Locator**), im ganzen Internet eindeutige Bezeichnung eines Dokuments.

- Eine URL hat folgende Form:

  `PROTOKOLL://RECHNERNAME/LOKALER_NAME`

  wobei `PROTOKOLL` das Verfahren zur Übertragung des Dokuments angibt, `RECHNERNAME` der Domainname des Servers ist, der das Dokument bereithält, und `LOKALER_NAME` Informationen über den Ort des Dokuments auf dem Server enthält.

- ☐ `http://www.harri-deutsch.de/verlag/titel/stoecker/s_1573.htm`

  - ▷ `LOKALER_NAME` ist im einfachsten Fall ein Verzeichnispfad zum gesuchten Dokument auf dem Server; es kann aber vom Server auch anders interpretiert werden. Der letzte Teil ist meistens ein Dateiname (`s_1573.html`). Wird er weggelassen, sucht der Server in der Regel nach einer Datei `index.html` oder `index.htm` an der entsprechenden Stelle.
  - ▷ http ist das Protokoll zur Übertragung von HTML-Dokumenten (**WWW-Seiten**). Andere Protokolle dienen hauptsächlich zur Integration weiterer Internet-Dienste.

- ☐ Weitere Protokolle:

Name	Bedeutung
ftp	Übertragung von Dateien
news	Anzeigen von Newsgruppen
file	Zugriff zu Dateien auf dem eigenen Rechner

- Wesentliche Funktion des Browsers ist die formatierte Darstellung von HTML-Dokumenten. Querverweise zu anderen Dokumenten (**Hyperlinks** oder einfach **Links**) werden besonders gekennzeichnet (z.B. farblich oder durch Unterstreichung). Durch einfaches Anklicken eines Links wird die entsprechende Seite geladen und angezeigt.
  - ▷ Browser bieten die Möglichlichkeit, zum letzten Dokument, meist auch zu allen bisherigen Dokumenten, zurückzugehen.
  - ▷ Eine URL kann auch direkt als Text angegeben werden. Darüberhinaus kann man eine Liste interessierender Seiten anlegen (**Bookmarks**, **Favoriten**) und in hierarchischer Form gliedern.

Homepage, Einstiegsseite zu den WWW-Seiten einer Institution, Firma oder Person.

- ▷ Eine gut gepflegte Liste von Bookmarks ist eine wesentliche Orientierungshilfe im unüberschaubaren Informationsangebot des WWW. Dabei empfiehlt es sich, auch Verweise auf übergeordnete Seiten wie Homepages aufzunehmen, da diese sich erfahrungsgemäß nicht so häufig ändern wie einzelne untergeordnete Seiten.

- WWW-Dokumente können auch multimediale Elemente enthalten.
  - ☐ Bilder, Sprache und Musik, Filme und Animationen, 3D-Darstellungen, eingebette Programme (Applets).
- Multimedia-Elemente sind mit Typ-Informationen (**MIME-Typ**) versehen, anhand derer der Browser entscheidet, wie er sie darstellen soll. Dabei hat er drei Möglichkeiten:
  1. Er kann sie selber direkt im Dokument anzeigen (einige Bildformate).
  2. Er kann sie mit Hilfe besonderer Erweiterungsmodule (**Plugins**) innerhalb des Dokuments anzeigen.

3. Er ruft externe Programme auf, die sie in einem eigenen Fenster darstellen.

▷ Plugins für die verbreiteten Browser werden von anderen Firmen - meist für die von ihnen entwickelten Formate - zur Verfügung gestellt. Für die häufigsten MIME-Typen werden die benötigten Plugins direkt mit den Browsern mitgeliefert.

▷ Die Verwendung von Formaten, die Plugins benötigen, schränken den Leserkreis einer Seite ein, da sie evtl. beim Benutzer nicht installiert sind oder für das verwendete Betriebssystem bzw. den benutzten Browser nicht zur Verfügung stehn.

□ Für die folgenden Formate sind Plugins weit verbreitet:

Format	Beschreibung
Wave	Audio-Clips
MIDI	Musikstücke
Quicktime	Videos, Animationen
RealAudio, RealVideo	Ton-, Filmübertragung in Echtzeit
VRML	dreidimensionale Darstellungen
PDF	strukturierte Text-Dokumente

● Um den Netzwerkverkehr und die Ladezeiten zum Betrachten eines Dokuments zu verringern, werden bereits geladene Seiten in Zwischenspeichern (**Caches**) aufbewahrt und bei Bedarf direkt von dort geholt. Dabei werden in der Regel drei verschieden große Cache-Stufen benutzt:

- im Hauptspeicher (**Speicher-Cache**)
- auf der Festplatte (**Disk-Cache**)
- auf externen Rechnern (**Proxy-Cache**)

▷ Proxy-Caches werden auf Maschinen des Providers eingerichtet (an Unis an den Rechenzentren), um Kopien häufig gelesener Seiten lokal vorzuhalten. Sie bilden untereinander ein Netzwerk, das eine irgendwo zwischengespeicherte Seite möglichst schnell heranschaffen soll.

▷ Bei Änderungen an bestehenden Seiten enthalten Proxies u.U. veraltete Versionen. Seiten mit häufigen Änderungen können ein *Verfallsdatum* angeben, nach dem sie neu vom Original-Server geholt werden sollen. Außerdem kann man den Browser anweisen, eine geladene Seite ohne Cache-Verwendung zu holen (bei Netscape z.B. mit <SHIFT>-Reload).

b) Email

Email, das Versenden von elektronischer Post, gehört zu den ältesten und verbreitetsten Internet-Diensten.

● Eine Email hat neben dem Inhalt auch einen **Header** mit zusätzlichen Informationen. Der Header besteht aus mehreren Zeilen, die jeweils mit einem Schlüsselwort beginnen.

□

Schlüsselwort	Bedeutung
`To:`	Adresse
`From:`	Absender
`Reply-To:`	Antwort-Adresse (u.U. verschieden von `From:`)
`Subject:`	einzeilige Kurzbeschreibung, Betreff
`Date:`	Datum und Uhrzeit des Versendens

● Wichtiger Bestandteil einer Email ist die Adresse. Sie hat die Form

```
NAME@DOMAINNAME
```

Dabei weist `NAME` auf die Person des Empfängers, `DOMAINNAME` auf den Rechner, der die Mail empfängt.

□ `Junglas@tu-harburg.de`

▷ Das `To:`-Feld einer Mail kann auch mehrere, durch Komma getrennte Adressen enthalten.

Mailingliste, Mailadresse, die einer ganzen Gruppe von Personen zugeordnet ist. Jede Mail an eine Mailingliste geht an alle Mitglieder. Mailinglisten werden vom Mailadministrator einer Domain (**Postmaster**) verwaltet.

▷ Eine Mailingliste ist ein (privater) Diskussionskreis. Viele Mailinglisten sind aber öffentlich, d.h. jeder kann durch eine einfache Mail automatisch Mitglied werden.

▷ Es gibt kein weltweites Email-Verzeichnis. Man findet zumindest die Adresse von Personen, von denen man Email bekommen hat oder News-Artikel liest, im Header. Email-Adressen von Uni-Angehörigen sind meistens unter der Homepage ihres Instituts zu finden.

● Der Inhalt einer Email besteht im einfachsten Fall aus einem ASCII-Text. Um auch andere Formate (Texte mit Umlauten, HTML-Texte, Bilder etc.) übertragen zu können, werden sie mit einem entsprechenden MIME-Typ gekennzeichnet. Darüberhinaus können einer Email noch beliebige weitere Dateien angehängt werden (**Attachments**).

▷ Weit verbreitet ist die Konvention, am Ende des Textes eine explizite Absender-Angabe unterzubringen (**Signature**). Sie enthält neben der Adresse oft noch die URL der Homepage oder die Telefonnummer.

● Neben den eigentlichen Empfängern kann man eine Email auch als Kopie nachrichtlich an weitere Personen senden. Dazu ist das Header-Feld **CC:** (**Carbon Copy**) vorgesehen.

▷ Alle Empfänger einer Mail sehen im Header die vollständigen `To`- und `CC`-Listen.

● Das Header-Feld **BCC:** (**Blind Carbon Copy**) dient, wie `CC`, zum Versenden von Kopien einer Email. Im Gegensatz zu `CC` sind die Empfänger einer `BCC`-Nachricht für die anderen Empfänger nicht zu sehen.

▷ Damit Emails zugestellt werden können, sollte der Mailserver einer Domain ständig mit dem Internet verbunden sein. Die Empfänger dagegen holen sich ihre Mails nach Bedarf vom Mailserver.

● Zum Transport von Mails vom Mailserver zum Empfänger gibt es zwei verschiedene Verfahren:

POP Mails werden vom Server zum Empfänger kopiert und auf dem Server gelöscht.

IMAP Mails können zusätzlich auf dem Mailserver gespeichert und organisiert werden.

▷ Zum Empfangen, Verschicken und Organisieren von Emails gibt es eine Reihe von Programmen mit einer Vielzahl an Funktionen.

□ Wichtige Zusatz-Funktionen von Mailprogrammen:

- Organisieren von Mails in Foldern (Verzeichnissen)
- Verwalten von Adressen
- **Reply**-Funktion: auf eine empfangene Mail antworten
- **Forward**-Funktion: eine eingegangene Mail weiterschicken

▷ Die Vertraulichkeit einer Email ist grundsätzlich nicht gewährleistet: Sowohl der Inhalt als auch Absender oder Empfänger einer Email können prinzipiell gelesen und geändert werden. Daher wurden Verfahren entwickelt, um die Richtigkeit des Absenders (**Authentisierung**) und des Inhalts (**Integrität**) einer Email zu gewährleisten. Sie basieren auf mehrstufigen Verschlüsselungsverfahren.

c) News

News, Diskussionsgruppen, in denen ähnlich zu schwarzen Brettern alle Beiträge für alle Leser einer Gruppe sichtbar sind.

● Die mehr als 10000 Newsgruppen sind nach ihrem Thema in Hierarchien eingeordnet und benannt.

□ `de.comp.os.linux`

deutsche Gruppe . Computer . Betriebssysteme (operating systems) . Linux

□ `sci.math.research`

Wissenschaft (science) . Mathematik . Forschung

□ Einige wichtige Hauptkategorien:

Kürzel	Bedeutung
alt	alternative Gruppen (weit gestreut)
comp	Computer
de	deutsche Gruppen
news	Informationen über Newsgruppen
rec	Hobbies (recreations)
sci	Wissenschaften
soc	soziale Themen

● Neue Newsgruppen entstehen, wenn sich genügend viele Interessenten finden. Der genaue Prozeß besteht in einem formalisierten Abstimmungsverfahren.

▷ Viele Newsgruppen werden weltweit im ganzen Internet verbreitet, andere nur innerhalb eines Landes oder einer Domain (etwa einer Universität).

● Es gibt moderierte Gruppen, in denen die Beiträge von einer Person, dem **Moderator**, vor der Veröffentlichung geprüft werden (auf Zugehörigkeit zum Thema der Gruppe, auf Relevanz, etc., je nach Gruppe). In unmoderierten Gruppen werden dagegen alle Beiträge, die Benutzer abgeschickt (**geposted**) haben, weiterverbreitet.

● Artikel in Newsgruppen werden von einem zentralen Server (**Newsserver**) einer Domain bezogen und zwischengespeichert. Aufgrund der ungeheuren Datenflut werden sie nach einer bestimmten Zeit (einige Tagen bis Wochen) auf dem Server wieder gelöscht.

Newsreader, Programm zum Lesen und Posten von News-Artikeln.

□ nn, tin, xrn, netscape

▷ Newsreader unterstützen verschiedene Aspekte der Arbeit mit News:

- Organisieren von Newsgroups:
  Auswahl der Gruppen, die man lesen möchte (**subscribe**-Funktion), Anzeigen neuer Gruppen
- Lesen von Artikeln:
  Sortieren von Artikeln nach Thema (Subject-Zeile), Anzeigen der Subject-Zeilen zur schnellen Übersicht, Markieren von Artikeln als gelesen /ungelesen
- Schreiben von Artikeln:
  Posten in eine oder mehrere Gruppen, Email an den Autor eines Artikels

▷ Häufig tauchen, vor allem von neuen Teilnehmern, immer wieder dieselben Fragen auf. Um ständige Wiederholungen zu vermeiden, werden in vielen Newsgruppen Listen mit den häufigsten Fragen und ihren Antworten (**FAQ-Listen** = Frequently Asked Questions) regelmäßig geposted.

▷ Um ein friedliches Miteinander der Teilnehmer zu gewährleisten, haben sich gewisse Regeln eingebürgert (als **Netiquette** bezeichnet), die sich wohl von selbst verstehen dürften, z.B.:

- Vorsicht mit Äußerungen über andere.
- Benutze informative Subject-Zeilen.
- Fasse Dich kurz.

d) Übertragung von Dateien

● Mit Hilfe von **ftp** (File Transfer Protocol) können Dateien zwischen Rechnern übertragen werden.

▷ Ftp wird im WWW hauptsächlich für Datensammlungen (etwa Programm- oder Artikel-Archive) eingesetzt.

- ▷ Häufig werden bestimmte Datei-Formate zum Zusammenfassen und Komprimieren der Daten benutzt.
- □ `zip`, `tar.gz`
- ● Mit einer entsprechenden URL kann man sich ganze Verzeichnisse ansehen oder einzelne Dateien direkt mit einem Browser auf den eigenen Rechner übertragen (**downloaden**).
- □ `ftp://ftp.zib.de/netlib/eispack`
  Inhaltsverzeichnis von Eispack (Software zur numerischen Lösung von Eigenwertproblemen)
- □ `ftp://ftp.zib.de/netlib/eispack/archives/eispack-linux.tgz`
  Komplette Bibliothek für Linux
- ▷ Spezielle FTP-Programme erlauben z.B. das Übertragen von ganzen Verzeichnissen.
- ▷ Auch alle im Browser angezeigten WWW-Seiten und die meisten Inhalte (Bilder, Applets) lassen sich direkt abspeichern.

e) Telnet

Telnet, Protokoll zum Anmelden (**Einloggen**) und Arbeiten an einem anderen Rechner, wobei ein einfaches Text-Terminal nachgebildet wird.

- ▷ Um sich auf einem anderen Rechner einloggen zu können, benötigt man dort eine Zugangsberechtigung. Man identifiziert sich mit einem Usernamen und einem Paßwort.
- ▷ Einige Internet-Dienste verwenden Telnet, wobei ein eingeschränkter Gast-Zugang zur Verfügung gestellt wird.
- □ Online-Kataloge (**OPACs**) von Bibliotheken, etwa

  `telnet://opac.tu-harburg.de`

- ▷ Benutzt man eine solche URL in einem Browser, öffnet dieser ein Fenster mit einem Text-Terminal. Es gibt aber auch spezielle Telnet-Programme mit zusätzlichen Funktionen (z.B. Mitschreiben in einer Protokolldatei).

Gezielte Informationssuche

Wenn man sich durch einige der größeren WWW-Server klickt, findet man Informationen zu allen Lebenslagen, von den neuesten Nachrichten über die wichtigsten Baseball-Ergebnisse zum aktuellen Horoskop. Nebenher wurde man mit ungezählten Werbebildern bombardiert. In diesem Wust gezielt Informationen zu finden, erfordert das Anwenden verschiedener Strategien, von denen die wichtigsten im folgenden vorgestellt werden.

- ▷ Homepages von Firmen oder Organisationen kann man häufig direkt erraten.
- □

Muster	Gruppe	Beispiel
`www.FIRMA.com`	Firmen (oft amerikanisch)	`www.compaq.com`
`www.FIRMA.de`	deutsche Firmen (auch unter .com)	`www.siemens.de`
`www.NAME.org`	Organisationen	`www.linux.org`
`www.uni-STADT.de`	deutsche Hochschulen	`www.uni-frankfurt.de`
`www.tu-STADT.de`		`www.tu-harburg.de`
`www.fh-STADT.de`		`www.fh-friedberg.de`

- ▷ Zu vielen speziellen Themen gibt es besondere Link-Sammlungen.
- □ Mehrere mathematische Institute in Deutschland stellen im Rahmen des MathNet-Projekts eine große Sammlung von Links zur Mathematik zusammen unter `http://www.math-net.de/`
- □ Die Computer-Zeitschrift c't unterhält eine Seite, die Hilfe zur Selbsthilfe bei Problemen mit PC-Hardware oder -Programmen bietet:

`http://www.heise.de/ct/tipsundtricks/`

▷ Bücher findet man in den Online-Katalogen (**OPACs**) der Bibliotheken.

□ Die deutsche Zentralbibliothek für u.a. Mathematik und Physik ist die TIB/UB an der Uni Hannover. Man kann in ihren umfangreichen Beständen suchen unter

`telnet://opac.tib.uni-hannover.de`

▷ Artikel, die in den News veröffentlicht wurden, werden seit 1995 auf dem Dejanews-Server gesammelt:

`http://www.dejanews.com/`

Hier kann man gezielt spezielle Gruppen nach Stichworten durchsuchen.

□ Suche nach dem Stichwort `Strassen` in der Gruppe `sci.math.research` liefert 6 Artikel, die sich alle mit dem Strassen-Algorithmus beschäftigen. Erweitert man die Suche auf alle Newsgruppen mit `math` im Namen (also `*math*`), erhält man 72 Antworten, von denen einige die Wahrscheinlichkeit diskutieren, mit gewissen "Strassen" beim Monopoly zu gewinnen. Natürlich ändern sich diese Zahlen häufig.

▷ Zur Orientierungshilfe wurden von verschiedenen Firmen große WWW-Server installiert, die einen möglichst großen Teil des Internets absuchen und in teils verschiedener Weise zum Abruf bereitstellen. Diese Dienste werden durch Werbung finanziert.

Web-Katalog, Suchsystem, bei dem die Informationen thematisch in Kategorien einsortiert werden. Dies geschieht i.a. manuell, d.h. durch Redakteure.

□

Name	URL	Bemerkungen
Dino	`www.dino-online.de`	deutsche Seiten, entstanden an der Uni Göttingen
Yahoo!	`www.yahoo.com`	i.w. angloamerikanisch, Klassiker
	`www.yahoo.de`	Yahoo!-Ableger, deutschsprachige Inhalte

▷ Web-Kataloge umfassen nur einen kleinen Teil des Internets, bieten dafür aber einen guten Überblick. Sie sind besonders nützlich für

- einen Überblick zu einem Thema
- weitgespannte Anfragen

□ Im Dino-Katalog finden wir unter "Wissenschaft", Unterpunkt "Mathematik+Statistik" z.B. eine Liste von mathematischen Instituten, die Homepage der Deutschen Mathematiker-Vereinigung oder eine spezielle Mathematik-Suchmaschine der FH Bochum.

Suchmaschine, Suchsystem, das automatisch Bereiche des Internets absucht und eine große Datenbank mit Schlagwörtern anlegt. Suchen im Datenbestand geschehen über eine einfache Oberfläche oder eine kompliziertere Abfragespache.

□

Name	URL	Bemerkungen
Altavista	`www.altavista.com`	erste GROSSE Suchmaschine
HotBot	`www.hotbot.com`	
Fireball	`www.fireball.de`	deutsche Dokumente, Abfragesprache wie bei Altavista

▷ Suchmaschinen schaffen heute etwa 10 - 40 % des gesamten WWW-Bestandes. Sie besuchen eine Seite im Abstand von einigen Wochen, nicht selten sind daher Links im Datenbestand veraltet oder nicht mehr existent.

▷ Suchmaschinen eignen sich besonders für

- Suchen nach Namen oder seltenen Begriffen
- durch mehrere Bedingungen eingrenzbare Abfragen

▷ Einfache Suchen nach einem einzigen Schlagwort liefern oft eine Unzahl an Treffern. Wichtig ist die geschickte Eingrenzung mit Hilfe der Abfragesprache.

▷ Die Abfragesprachen der einzelnen Suchmaschinen unterscheiden sich. Für gezielte effiziente Suchen sollte man sich mit den Abfragesprachen einiger großer Suchmaschinen vertraut machen.

☐ Einige Elemente der Abfragesprache von Altavista:

Syntax	Bedeutung
`allesklein`	paßt auf Klein- und Großbuchstaben, etwa `allesklein`, `AllesKleiN`
`kleinUndGroß`	paßt nur auf genau die angegebene Form
`WORT1 WORT2 ...`	mindestens eins der Worte muß auftreten, sortiert wird nach Anzahl der Treffer
`"WORT1 WORT2"`	Der angegebene Text muß genau so auftreten (bis auf Klein-/Großschreibung)
`+WORT1 +WORT2 -WORT3`	`WORT1` und `WORT2` müssen auftreten, `WORT3` darf nicht auftreten
`do*ument`	`*` als Platzhalter (**Wildcard**) für irgendwelche Zeichen, paßt etwa auf `document`, `dokument`

☐ Suche in Altavista:

Suchbegriff	Ergebnis
`Strassen`	89540 Einträge, fast alle über (Auto-)Straßen
`+Strassen +matrix`	3100 Einträge
`+Strassen +matrix +implementation`	196 Einträge

▷ Kataloge und Suchmaschinen werden auch kombiniert.

☐ Wird ein Suchbegriff in den Yahoo-Seiten nicht gefunden, kann er gleich an eine Suchmaschine weitergegeben werden.

▷ Mit der steigenden Zahl der Suchmaschinen und Kataloge - z.Zt. schon über 100 -, die alle nur einen Teil des Internets abdecken, wird selbst die Suche in Suchmaschinen recht aufwendig.

Meta-Suchmaschine, Suchsystem, das eine Anfrage gleichzeitig an mehrere Suchmaschinen weitergibt, die innerhalb einer bestimmten Zeit angekommenen Antworten zusammenfaßt und ggf. bewertet.

▷ Meta-Suchmaschinen erlauben nicht die volle Ausnutzung von Abfragesprachen der einzelnen Suchmaschinen. Bestenfalls haben sie eine eigene Syntax, die sie in die der jeweiligen Suchmaschine umsetzen.

☐

Name	URL	Bemerkungen
MetaCrawler	`www.metacrawler.com`	eigene Eingabesyntax
Highway 61	`www.highway61.com`	erlaubt Weitersuchen
MetaGer	`meta.rrzn.uni-hannover.de`	deutsche Suchmaschinen

▷ Ziel einer Suche sollte in der Regel nicht nur sein, irgendein Dokument zu finden, sondern eine gute Einstiegsseite zum interessierenden Thema. Eine gut gepflegte Sammlung von URLs wird so im Lauf der Zeit die beste, individuelle “Suchmaschine”.

HTML

HTML (**Hypertext Markup Language**), die Sprache des WWW, dient zur Beschreibung der Struktur eines Dokuments, erlaubt die Verknüpfung von Dokumenten aller Art mittels **Hyperlinks** und das Einbinden vielfältiger multimedialer Elemente.

- Grundprinzip von HTML ist die Trennung von Struktur und Layout: HTML dient zur Beschreibung von Inhalt und logischer Struktur eines Dokuments. Die genaue Darstellung (Wahl der Fonts, Zeilenabstände etc.) wird grundsätzlich dem Browser überlassen.
  - ▷ Der Wunsch nach genauerer Festlegung des Layouts führte zunehmend zu HTML-Erweiterungen, die dieses Prinzip durchbrachen. Mit der HTML-Version 4.0 wurden alle die Darstellung betreffenden Elemente entfernt; zur Festlegung des Layouts dienen nun besondere Sprachelemente bzw. Beschreibungsdateien, die **Style Sheets**. Übergangsweise gibt es eine HTML-Version (4.0 Transitional), die zusätzlich noch die meisten der früheren Elemente unterstützt.
  - ▷ HTML-Seiten bestehen aus reinen Texten und können mit beliebigen Editoren erstellt werden. Es gibt aber auch graphische HTML-Editoren, die ähnlich wie ein Textverarbeitungsprogramm arbeiten und den eigentlichen HTML-Code selbständig erzeugen.
- Eine HTML-Seite besteht aus dem eigentlichen Text, in den HTML-Elemente (**Tags**) zur Beschreibung der Struktur eingebettet sind.
- Der Text kann aus allen Zeichen des internationalen Unicode-Zeichensatzes bestehen. In der Regel wird man sich aber auf eine Teilmenge beschränken (z.B. den Latin-1-Zeichensatz mit den meisten westeuropäischen Umlauten oder sogar nur ASCII), wie sie etwa durch die eigene Tastatur nahegelegt wird. Zusätzliche Zeichen (Umlaute, spezielle Sonderzeichen) können dann durch einen numerischen Code als `&#NNN;` oder einen besonderen Namen in der Form `&NAME;` eingegeben werden.

□

Sonderzeichen	HTML-Bezeichnung
Ä Ö Ü	`&Auml; &Ouml; &Uuml;`
ä ö ü ß	`&auml; &ouml; &uuml; &szlig;`
æ å é	`&aelig; &aring; &eacute;`
< > & "	`< > & &quot;`

- Zeilenumbrüche und zusätzliche Leerzeichen werden vom WWW-Browser ignoriert. Die Zeilen werden - wie in Textverarbeitungsprogrammen - automatisch umgebrochen.
- HTML-Tags weisen einem Text ein bestimmtes strukturelles Element zu. Sie werden in spitze Klammern eingeschlossen und umschließen in der Regel den Text in der Form

```
<TAG>text text text</TAG>
```

  Der Tag-Name kann klein oder groß geschrieben werden.

□

Element	Bedeutung
`<H1>Einleitung</H1>`	Hauptüberschrift
`<em>Beachte:</em>`	betonter Text
`A<SUB>ij</SUB>`	tiefergestellter Text

- Bei einigen Elementen ist das Endtag überflüssig und kann entfallen.

□

Element	Bedeutung
`<P>`	Absatz
`<LI>`	Element einer Liste

- Manche Elemente haben keinen Inhaltstext, auch bei ihnen entfällt das Endtag.

□

Element	Bedeutung
` `	neue Zeile
`<HR>`	horizontale Linie

- Tags können zusätzliche Eigenschaften (**Attribute**) haben, denen in der Regel Werte zugewiesen werden. Sie werden angegeben als

```
<TAG ATTRIBUT1="wert1" ATTRIBUT2="wert2" ...>
```

□ `<IMG src="blume.gif" alt="Blume">`

Das `IMG`-Tag fügt in den Text das durch das Attribut `src` bezeichnetet Bild ein. Das Attribut `alt` enthält einen Text, der statt des Bildes angezeigt werden kann (z.B. bei reinen Textbrowsern).

● HTML-Dokumente können Kommentare enthalten, die vom Browser nicht angezeigt werden. Sie werden in <`!--` und `-->` eingeschlossen.

□
```
<!-- Dies ist ein Kommentar
       ueber zwei Zeilen  -->
```

● Eine HTML-Seite besteht aus einem Kopfteil (**Header**), der den Titel der Seite und zusätzliche Angaben (Author, Stichworte für Suchmaschinen etc.) enthalten kann, und dem Hauptteil (**Body**) mit dem gesamten sichtbaren Inhalt. Beide Teile werden vom <`HTML`>-Tag umschlossen:

```
<HTML>
<HEAD>
  <TITLE>Der Titel des Dokuments</TITLE>
</HEAD>
<BODY>
  Der Inhalt des Dokuments
</BODY>
</HTML>
```

▷ Der Titel wird in der Regel vom Browser in der Titelzeile des Fensters angezeigt.

□ Die folgenden Tabelle zeigt die wichtigsten HTML-Tags zur Strukturierung von Texten. Die Spalte “Darstellung” gibt an, wie die entsprechenden Elemente im Browser typischerweise dargestellt werden.

Tag	Bedeutung	Darstellung
<`H1`>	Haupt-Überschrift	eigener Absatz, sehr großer Font, fett
<`H2`> ... <`H6`>	Unter-Überschriften	eigener Absatz, großer Font, fett
<`P`>	Absatz	neue Zeile, größerer Zeilenabstand
<`BR`>	Zeilenumbruch	neue Zeile
<`UL`>	Aufzählungsliste	eigener Absatz, eingerückt
<`LI`>	Listenelement	eigene Zeile pro Element, beginnt mit Spiegelstrich (Kreis, Quadrat etc.)
<`OL`>	numerierte Liste	wie <`UL`>, aber Elemente werden durchnumeriert
<`PRE`>	vorformatierter Text	nicht-proportionaler Font, Zeilenumbrüche und Leerzeichen werden beibehalten
<`EM`>	betonter Text	schräggestellt
<`STRONG`>	stark betonter Text	fett

▷ Listen können beliebig geschachtelt werden.

● Verweise (Links) auf andere Dokumente werden mit dem <`A`>-Tag gekennzeichnet:

```
<a href="URL">Verweistext</a>
```

Dabei gibt `URL` das Ziel des Links an, `Verweistext` den Text im Dokument, der als Link gekennzeichnet ist.

□ `URL` kann direkt eine Datei bezeichnen (`../inhalt.html`) oder eine beliebige WWW-Adresse (`http://...`, `ftp://...`).

- ▷ Browser zeigen den Linktext meistens unterstrichen und in einer anderen Farbe an. Wurde der Link bereits verfolgt, wird der Linktext in einer anderen Farbe dargestellt.
- ▷ Anstelle des Verweistextes kann auch ein Bild stehen. Browser zeigen einen solchen Link durch einen farbigen Rahmen um das Bild an oder durch eine Änderung des Mauszeigers, etwa in eine Hand.

● Grafiken können direkt in den Text eingebunden werden mit dem Tag

```
<IMG src="URL" alt="Text">
```

- ▷ Von den zahlreichen verbreiteten Bildformaten unterstützen alle gängigen Browser zumindest GIF und JPEG. Andere Formate erfordern ggf. Plugins zur Darstellung.
- ▷ Die Möglichkeit (z.B. bei GIF-Bildern), eine Farbe als "transparent" zu kennzeichnen, ermöglicht eine nahtlose Einfügung von nicht-rechteckigen Bildern.
- ▷ Solange noch keine verbreitete Möglichkeit zur direkten Darstellung mathematischer Formeln in einem HTML-Dokument besteht, kann man sie (z.B. mit Hilfe von LaTeX) als GIF-Bilder mit transparentem Hintergrund einfügen.
- ▷ Das GIF-Format erlaubt, mehrere Bilder zu einer kleinen Animation in einer GIF-Datei zusammenzufügen (**animated GIF**). Solche Animationen werden genau wie einfache GIF-Dateien mit dem IMG-Tag eingebunden.

● In eine HTML-Seite können in Java geschriebene Programme (**Applets**) eingebunden werden mit dem Tag

```
<applet code="Programmdatei"  width="WWW" height="HHH">
</applet>
```

Der Browser weist dem Applet, ähnlich wie einer Grafik, einen rechteckigen Bereich der Breite `WWW` und Höhe `HHH` in Pixeln zu, der zur Programmaus- und eingabe dient.

- ▷ Java-Applets müssen compiliert werden. Der compilierte Code liegt in einer Datei, die normalerweise die Endung `.class` trägt.
- ▷ Die folgende Tabelle führt einige weiterführenden Möglichkeiten von HTML auf. Zu deren genaueren Beschreibung sei auf die umfangreiche Literatur bzw. die vielen im WWW verfügbaren Kurse verwiesen. *

Element	Funktion
Tabelle	Darstellung von Elementen in Zeilen und Spalten
Imagemap	Links, die mit ausgewählten Bereichen einer Graphik verknüpft sind
Frame	Aufteilung einer Seite in Teilseiten
Style Sheet	Spezifikation des Layouts
Formular	Komponenten zur Eingabe von Daten
JavaScript	Scriptsprache für dynamische HTML-Seiten

* s. etwa http://www.teamone.de/selfaktuell/

24.5 Anwendungssysteme

Computeralgebra

numerische Werkzeuge

Sachwortverzeichnis

ablauffähiges Programm, 8
absoluter Pfad, 3
abstract, 32
abstrakte Basisklasse, 16
abweisende Schleife, 10, 25, 53
Adapterklasse, 39
aktives Fenster, 3
aktuelles Verzeichnis, 3
Algorithmus, 16
animated GIF, 70
Anweisung, 10
Applet, 22, 41, 70
Array, 11, 24, 49
assoziatives Array, 37
assumed shape, 55
Attachment, 63
Ausnahme, 27
Authentisierung, 63
automatisches Array, 56

Batchdatei, 4
Baum, 18
BCC:, 63
bedingte Anweisung, 10
Benutzeroberfläche, 3
Betriebssystem, 2
Bibliothek, 8
Blind Carbon Copy, 63
Block, 10, 25
Body, 69
Bookmark, 61
boolescher Wert, 11
break, 26
Browser, 60
Bytecode, 8

Cache, 62
call-by-reference, 12
call-by-value, 12, 28
Carbon Copy, 63
cast, 23
CC:, 63
CDE, 6
class, 24
Code-Browser, 9
Collection-Klasse, 37
Common Desktop Environment, 6
Compiler, 8
Component, 41
Content Pane, 42
continue, 26

Datei, 2
Dateisystem, 2
Datentyp, 11
Debugger, 9
Desktop, 4, 6
Directory, 3
Disk-Cache, 62
Divide and Conquer, 19
DO-Schleife, 11
Domain-Name, 59
doppelt verkettete Liste, 18
Download, 65
dynamische Datenstruktur, 18

einfach verkette Liste, 18
einfacher Datentyp, 11, 23, 48
Einloggen, 65
Elternverzeichnis, 3
Email, 62
Event, 39
Exception, 27
Executable, 8
Explorer, 4
extends, 30

FAQ-Liste, 64
Favorit, 61
Feld, 11, 24, 49
Fenster, 3
FFT, 20
File, 2
Filesystem, 2
Filter Stream, 36
final, 31
for-Schleife, 11
Forward, 63
Frame, 39
ftp, 64
Funktion, 13

garbage collection, 14, 24, 28

globale Variable, 13
Graphical User Interface, 3
Graphics, 39
graphische Benutzeroberfläche, 3
GUI, 3, 41

Header, 62, 69
Homepage, 61
HTML, 60, 67
http, 60
Hyperlink, 61, 67
Hypertext Markup Language, 67
Hypertext Transfer Protocol, 60

Icon, 4
if-else, 25, 52
IMAP, 63
implements, 33
implizite do-Schleife, 58
import, 33
innere Klasse, 29
Input Stream, 36
integrierte Entwicklungsumgebung, 9
Integrität, 63
intent, 55
Interface, 33
Internet, 59
Internet-Dienst, 59
Interpreter, 8
IP-Adresse, 59
Iterator, 38

Java Development Kit, 22

KDE, 6
Klasse, 14, 28
Klassen-Browser, 9
Klassenbibliothek, 14
Klassenvariable, 30
Konstruktor, 28

Label, 26
late binding, 16
Laufanweisung, 11, 26, 54
LayoutManager, 41
Link, 61
Linker, 8
Listener, 39
listengesteuerte Ein-/Ausgabe, 57
Loader, 8
Lock, 46

logischer Wert, 11
lokale Variable, 12
Lynx, 60

Mailingliste, 63
Makefile, 9
Map, 37
Maschinensprache, 8
Math, 35
Mehrfach-Vererbung, 15
Mehrgitter-Verfahren, 20
Meta-Suchmaschine, 67
Methode, 14, 22
MIME-Typ, 61
Moderator, 64
Modul, 56
mounten, 3
Multi-Tasking-Betriebssystem, 2
Multi-User-Betriebssystem, 2
Multigrid-Verfahren, 20

Netiquette, 64
new, 23
News, 63
Newsreader, 64
Newsserver, 64
nichtabweisende Schleife, 10, 26, 53

Object, 31
Objectcode, 8
Objekt, 14, 28
objekt-orientierte Programmiersprache, 13
OPAC, 65, 66
Optimierung, 8
Output Stream, 36
Overhead, 9

package, 33, 34
Paket, 33
Parameterliste, 12
Plugin, 61
Pointer, 11, 51
Polymorphie, 16, 32
POP, 63
portabel, 8
posten, 64
Postmaster, 63
private, 29
Processing Stream, 36
Profiler, 9
Programmeinheit, 54

protected, 31
Protokoll, 59
Proxy-Cache, 62
public, 29

Quellcode, 8
quicksort, 17

Rapid Prototyping, 8
Record, 11, 51
rekursive Funktion, 13
relativer Pfad, 3
repeat-Schleife, 10
Reply, 63
Rootverzeichnis, 3

Schleife, 10, 53
Schnittstelle, 14, 33
Shell, 4
Shellskript, 4
Signature, 63
Single-Tasking-Betriebssystem, 2
Single-User-Betriebssystem, 2
Speicher-Cache, 62
Standard-Konstruktor, 29
Standardausgabe, 5
Start-Menü, 4
statische Variable, 30
Strassen-Algorithmus, 19
Stream, 36
String, 11, 24, 34, 51
StringBuffer, 35
Struktur, 11, 51
strukturierte Programmierung, 10
Style Sheet, 68
subscribe, 64
Suchmaschine, 66
super, 31
Synchronisation, 46
synchronized, 46
Syntaxfehler, 8
System, 35

Tag, 68
Tag-Attribut, 68
Task-Leiste, 4
TCP/IP, 59
Telnet, 65
this, 29
Thread, 44
throw, 27
Top-down-Entwurf, 13
Top-Level-Domain, 59

Überladen von Methoden, 15, 29
Uniform Resource Locator, 61
UNIX, 5
Unterprogramm, 12
Unterverzeichnis, 3
URL, 61

Variablendeklaration, 49
Vererbung, 15, 30
Verklemmung, 47
Verzeichnis, 3
Verzeichnisbaum, 3
Verzweigung, 10, 25, 53
virtuelle Java-Maschine, 22

Web-Katalog, 66
while-Schleife, 10
Wildcard, 5, 7, 67
Windows, 4
Windows 95, 4
Windows 98, 4
Windows NT, 4
World Wide Web, 60
Wurzelverzeichnis, 3
WWW, 60
WWW-Seite, 61
WWW-Server, 60

X-Windows, 5
X11, 5

Zeichenkette, 11, 24, 51
Zeiger, 11, 51
Zeigerarithmetik, 12
Zerlegungs-Verfahren, 19
Zustand, 14