

Java J1

1. Algorithmus

Ein **Algorithmus** ist ein Verfahren, das den Weg zur Lösung von Problemen beschreibt. Die Befolgung dieser Handlungsvorschrift garantiert das Erreichen des gewünschten Ziels. Ein Algorithmus muss folgende Eigenschaften haben:

- ❖ **Eindeutigkeit:** Die Vorschrift kann in einzelne, eindeutig ausführbare Schritte zerlegt werden. Die Reihenfolge der Schritte ist eindeutig.
- ❖ **Endlichkeit:** Das Verfahren endet.
- ❖ **Allgemeingültigkeit:** Die Vorschrift ermöglicht das Lösen einer Klasse von Problemen nach dem gleichen Schema (das unterscheidet einen Algorithmus von einem Rezept).



Ein Kochrezept (eventuell mit Zutaten als Datenmaterial) ist eine zu spezielle Vorstufe.

"500g Mehl" – welches Mehl? Weizen-, Roggen- oder Dinkelmehl?
 → Ein Algorithmus muss eindeutig formuliert sein.

"Fülle die Zutaten in eine Schüssel." – Was kommt zuerst?
 → Hier ist die Reihenfolge der Schritte nicht eindeutig.

"Schlage das Eiweiß bis es steif ist." – Was ist, wenn Eigelb hineingeraten ist, und das Eiweiß nie steif wird?
 → Hier ist die 'Endlichkeit' nicht gewährleistet.

Geschichte

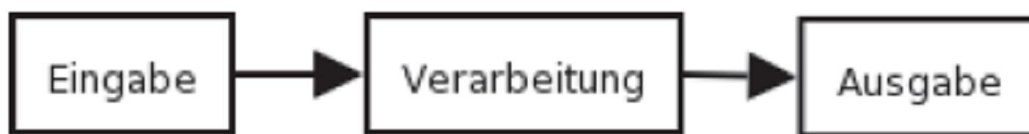
eine Version: Abu Ja'far Mohammed Ibn Musa al-Khowarizm (ca.780-850, Persien) schrieb das berühmte Lehrbuch 'Kitab al jabr w'al-muqabala' (Buch von der Ergänzung und Gegenüberstellung) über das Lösen von Gleichungen. Aus dem letzten Teil des Namens (eigentlich eine Ortsangabe) entstand das Wort Algorithmus, aus "al-jabr" entstand das Wort Algebra.

eine andere Version: Muhammad Ibn Musa Al-Chwarizmi (783-850, Bagdad) überträgt im 'Buch über die indische Rechnung' das indische Stellenwertsystem ins Arabische. Die Übersetzung des Buches ins Lateinische (13.Jhdt.) beginnt mit DIXIT ALGORIZMI.

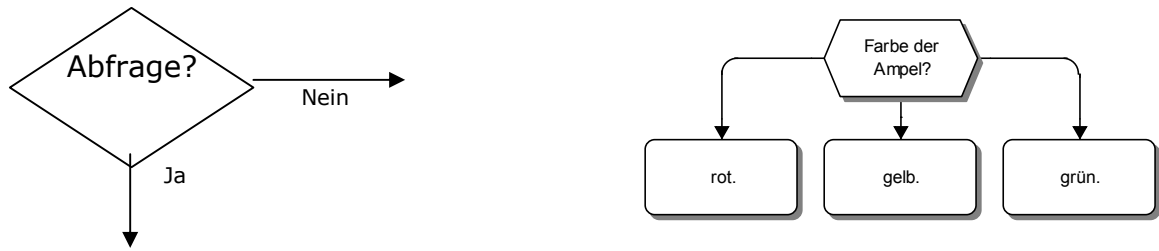
Grundstrukturen

Zum Entwurf von Algorithmen genügen drei Grundstrukturen. Diese lassen sich in Computerprogramme umsetzen.

Folge	Sequenz	<ul style="list-style-type: none"> ❖ Zu einem bestimmten Zeitpunkt wird nur eine Aktion durchgeführt. ❖ Jede Aktion wird genau einmal ausgeführt, keine wird wiederholt oder ausgelassen. ❖ Die Aktionen werden in der Reihenfolge ausgeführt, in der sie niedergeschrieben sind.
Auswahl	Selektion	Verzweigungen, Fallunterscheidungen
Wiederholung	Iteration	Schleifen



Fluss-Diagramme



Flussdiagramme können zur Darstellung von Algorithmen verwendet werden. Es werden inzwischen jedoch andere Darstellungsformen bevorzugt.

Struktogramme

aus Wikipedia

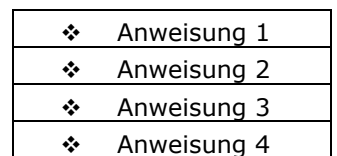
Ein **Nassi-Shneiderman-Diagramm (NS-Diagramm)** ist eine Entwurfsmethode für die strukturierte Programmierung. Es ist genormt nach DIN 66261. Benannt wurde es nach seinen Vätern, Dr. Ike Nassi und Dr. Ben Shneiderman. Da NS-Diagramme Programmstrukturen darstellen, werden sie auch als *Struktogramme* bezeichnet.

Die Methode zerlegt das Gesamtproblem, das man mit dem gewünschten Algorithmus lösen will, in immer kleinere Teilprobleme bis schließlich nur noch elementare Grundstrukturen wie Sequenzen und Kontrollstrukturen zur Lösung des Problems übrig bleiben. Diese können dann durch ein NS-Diagramm oder einen Programmablaufplan (PAP) visualisiert werden. Böhm und Jacopini haben 1966 nachgewiesen, dass sich jeder beliebige Algorithmus ohne Sprung (Goto) formulieren lässt.

Folge (Sequenz)

- ❖ Jede Anweisung wird in einen rechteckigen Strukturblock geschrieben.
- ❖ Die Strukturblocke werden nacheinander von oben nach unten durchlaufen.
- ❖ Leere Strukturblocke sind nur in Verzweigungen zulässig.

Struktogramm eines linearen Programms



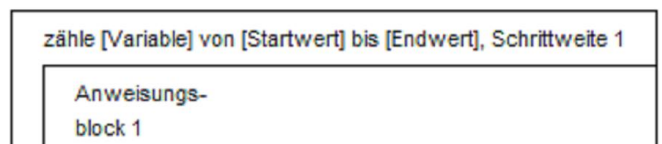
Auswahl (Selektion)

Beispiel



Wiederholung (Iteration)

Beispiel



Übungsaufgaben I

Algorithmen im Alltag und in der Informatik

1. Wörter suchen

In solchen Rätseln sind in einem oft quadratischen Feld Wörter versteckt, die horizontal, vertikal oder diagonal (auch rückwärts) geschrieben sind. In einem kleinen Feld lassen sich gesuchte Wörter noch schnell selbst finden. Schon bei 50x20 Zeichen ist es aber sehr ermüdend.

- Erläutere präzise, wie Du einen Suchknecht instruierst, damit er alle versteckten Wörter findet. Versteckt wurden fünf Wörter: Leim, Informatik, Abi, Kurs, Eimer
- Wie stellst Du Dir den Suchknecht vor?
- Könntest Du die fünf Worte auch finden lassen, wenn nur bekannt ist, dass irgendwelche deutschen Worte versteckt sind?

ASDFERFGTHZJUKIJERTX
 TIERNKMENOAERSOIMMEL
 KJHZRUINNVMGSHBOLE
 OKDFJRUEHZCCRBOHDARI
 IRGENSJISEREMIELIERN
 LPOUJNASHZRTEUFIFNEF
 EHSZEURKITAMROFNIAF
 AGREHZTUECJNBHEEIOUW

2. Sprecher suchen

- Für eine große Gruppe von Leuten ist ein Sprecher zu benennen. Die Gruppe einigt sich darauf, den Ältesten mit dieser Aufgabe zu betrauen. Erläutere schriftlich, wie sie den ältesten ermittelt.
- Lässt sich Dein Lösungsverfahren auch nutzen, um die größte von mehreren Zahlen zu bestimmen?

3. Wahl-Aufgabe

Bei Wahlen müssen die Stimmzahlen anschließend in Sitze umgerechnet werden.

- Beschreibe das Verfahren, nach dem diese Mandatsverteilung bei der Bundestagswahl geschieht. Suche geeignete Quellen!
- Bei einer Gemeinderatswahl am 17.8.2009 wurden für die sechs angetretenen Parteien folgende Stimmzahlen abgegeben:

A	B	C	D	E	F
20651	21716	8414	5304	4351	47848

- Welche Sitzverteilung ergibt sich nach dem oben beschriebenen Verfahren bei 22 zu vergebenden Sitzen (ohne Überhangsmandate)?

4. Schemen und Regeln

Viele Vorgänge des täglichen Lebens sind schematisiert und folgen festen Regeln; aus Gewohnheit und Bequemlichkeit, wegen Gesetzen und Geboten oder aus innerer Notwendigkeit. Diese Vorgänge ähneln damit Algorithmen.

- Beschreibe möglichst präzise die Alltagsalgorithmen:
 Herstellen einer Telefonverbindung / Schalten vom 1. in den 2. Gang / Binden beider Schuhe /
 Suchen eines Begriffs in Wikipedia oder einem Lexikon / Kochen zweier Frühstückseier
- Woran scheitert die Präzision z.B. im letzten Fall?

5. Verschlüsselung

Du tauschst häufiger Mitteilungen aus z.B. über das Handy. Leider können diese kurzen Botschaften auch von anderen Familienmitgliedern oder Freunden gelesen werden, wenn sie gerade das Handy haben. Du benutzt daher eine Geheimschrift. Eine einfache Vorschrift für Verschlüsselung von Texten könnte wie nebenstehend lauten.

- Verschlüssele damit: 'Ein erster, kurzer Test.'
- Welche Schwächen hat das Verfahren? Gib ein anderes Verfahren an. Welche Eigenschaften sollte das Verfahren besitzen?

Cäsar-Verschlüsselung:

Unterscheide nicht zwischen Groß- und Kleinschreibung; Beginne mit dem ersten Zeichen des Textes;

solange noch Zeichen da sind **wiederhole:**

wenn es ein Buchstabe außer 'z' ist,

dann ersetze ihn durch den alphabetisch nächsten,

sonst

wenn es ein 'z' ist,

dann ersetze ihn durch 'a';

wenn es ein Satzzeichen **oder** Leerzeichen ist,

dann übergehe es.

ende-solange

6. Leerstellen

- In einem Text sollen alle doppelten (mehrfachen) Leerstellen durch einfache Leerstellen ersetzt werden. Wie geht das?
- Was setzt Du bei dem voraus, der Dein Verfahren ausführt?

Schüler-Lösungen I

1. a)
 1. Nimm den ersten Buchstaben des ersten Wortes
 2. Suche ihn im Feld
 3. Hast du ihn gefunden, gehe in alle 8 Richtungen und schaue, ob sich das bekannte Wort finden lässt
 4. Findest du das Wort nicht, versuche es mit den restlichen Buchstaben, die noch im Feld sind
 5. Findest du das Wort, dann verfähre mit den restlichen Wörtern genauso
 6. Hast du alle Wörter gefunden, höre auf
- b)
 1. Nimm eine aktuelle Ausgabe des Dudens
 2. Gehe alle Wörter des Wörterbuchs durch und schaue, ob sie im Feld sind (Das selbe Verfahren wie bei a))
 3. Hast du fünf Wörter gefunden, höre auf
- a) Ich würde ihn zuerst nach dem Anfangsbuchstaben des ersten Wortes suchen lassen, danach in alle möglichen Richtungen nach dem zweiten, dann nach dem dritten usw.
- b) Ja, man müsste nur programmieren das der Computer jedes mögliche Wort nimmt, welches es in dem Quadrat gibt und mit einem Wörterbuch vergleichen lassen ob es dieses Wort im Deutschen gibt.
- a) Wie soll der Suchknecht funktionieren?
 - ❖ Der Suchknecht muss den ersten Buchstaben des Wortes kennen. Nun muss er jeden dieser Buchstaben in dem Rätsel heraussuchen. Danach gibt man ihm den zweiten Buchstaben.
 - ❖ Jetzt muss er rund um den ersten dieses Zweiten suchen (allerdings muss man noch hinzufügen dass die Wörter in einer Linie stehen...also nur vertikal, senkrecht oder diagonal, keine Knicke in einem Wort).
 - ❖ Genauso wird weitergemacht bis er das gewünschte Wort hat.
- b) Wörter finden lassen wenn nur bekannt ist, dass deutsche darin zu finden sind:
Der Suchknecht müsste jedes deutsche Wort kennen bzw. jede mögliche Buchstabenfolge mit einem Duden vergleichen. Die letztere Möglichkeit müsste die einfachere sein.

2. Die Leute sollen sich nach Jahrgang aufstellen. Danach soll der älteste Jahrgang sich nach Monaten ordnen. Die, deren Geburtsmonat am nächsten am Monat Januar ist, oder die im Januar Geburtstag haben, sollen wiederum das genaue Datum vergleichen. Die mit der niedrigsten Zahl müssen noch die Geburtsuhrzeit vergleichen. Wer früher geboren ist, ist der älteste und wird deshalb Sprecher.

4. a) Suchen eines Begriffs im Lexikon
Man nimmt das Lexikon in die Hand und schlägt das Buch auf die Seite 1 auf. Dann schaut man den Anfangsbuchstaben seines Wortes an und schaut ob das Buch an der Seite anzeigt, wo aufgeschlagen werden muss, um seinen Anfangsbuchstaben zu finden. Falls nicht blättert man so lange Blatt für Blatt um bis die fettgedruckten Wörter auf der Seite mit diesem Anfangsbuchstaben anfangen. Hat man die erste Seite auf dem die fettgedruckten Wörter mit dem Anfangsbuchstaben anfangen, schaut man, welches der zweite Buchstabe seines Wortes ist. Falls der zweite Buchstabe seines Wortes mit keinem zweiten Buchstaben der fg. Wörter auf der Seite übereinstimmt oder das Wort nur aus einem Buchstaben besteht, dann blättert, wie zuvor bis zu der Seite auf dem das erste Mal der zweite Buchstabe seines eigenen Wort und der zweite Buchstabe eines der fg. Wörter auf der Seite übereinstimmt. Wie mit dem zweiten Buchstaben, so verfährt man auch mit dem dritten, vierten und mit allen Buchstaben seines Wortes. Stimmen dann alle Buchstaben seines Wortes und einem der fg. Wörter auf dieser Seite überein liest man von diesem fg. Wort, bis zum nächsten fettgedrucktem Wort alles durch. *f.g.=fettgedruckt.

5. a) „Ein erster, kurzer Test“ wird zu: fjofstfslvsafsuftu
Es fällt auf, dass der Buchstabe „f“ sehr häufig vorkommt. Es liegt also nahe, dass dieser Buchstabe einen Vokal (außer u) ersetzt. Man braucht also nur die 5 Vokale durchprobieren und kommt schnell auf die einfache Verschlüsselung.

b) Andere Schlüssel (Siehe Dan Brown „ Diabolus“)

Der Klartext muss genau so viele Wörter enthalten wie eine Quadratzahl (49, 64, 81...).

Bei dem verschlüsselten Text müssen die Wörter gezählt werden.

Die Buchstaben werden nun in Reihen untereinander geschrieben. Die Länge einer Reihe wird durch die Wurzel der Wortanzahl bestimmt. Es wird nun von oben nach unten gelesen, dabei fängt man links an.

Beispiel:

DSSSAEICETUMEHLASBVLUMEUEÜNMICREGTNH = Die Verschlüsselung stammt aus einem Buch

D	S	S	S	A	E
I	C	E	T	U	M
E	H	L	A	S	B
V	L	U	M	E	U
E	Ü	N	M	I	C
R	S	G	T	N	H

Eigenschaften

Die Verschlüsselung sollte nicht sofort durchsichtig sein.

Es sollte eindeutig sein, damit beim Empfänger auch die richtige Nachricht ankommt.

Es sollte bei nicht PC-basierten Verschlüsselungen nicht zu lange dauern und umständlich sein zu ver- und entschlüsseln.

2. Programmiersprachen

Die eigentliche Sprache, die der Prozessor versteht und in der die Programme im Speicher vorliegen, nennt man **Maschinensprache**. Man kann einen Rechner direkt mit einer Bitfolge programmieren, denn genau so arbeitet er Befehle ab:

10100001 00000000 00101010 10001011 11011000 usw.

Diese Schreibweise nimmt zu viel Platz weg, man schreibt solche Daten hexagesimal auf.

Unser Maschinencode sieht dann so aus:

A1 00 2A 8B D8 C1 E0 02 03 C3 40 A3 00 28

Diese Maschinenbefehle stehen dann im ausführbaren Programm, z.B. als .EXE-Datei. Zur Ausführung werden sie in den Speicher gebracht (geladen) und der Prozessor holt sich die Maschinenbefehle nacheinander aus dem Speicher.

Assembler vereinfacht diese Eingaben, es wird dann direkt in Maschinensprache umgewandelt:

Assemblerbefehle	Daraus erzeugter Maschinencode
mov ax,B	A1 00 2A
mov bx,ax	8B D8
shl ax,2	C1 E0 02
add ax,bx	03 C3
inc ax	40
mov A,ax	A3 00 28

Solche Programme sind immer noch unübersichtlich. Die Maschinensprache ist außerdem vom Prozessor abhängig (nicht 'portabel').

Es gibt nun zwei Arten, die Maschinensprache zu umgehen: **Compiler** übersetzen eine verständlichere Sprache in die Maschinen-Anweisungen, die dann das ausführbare Programm bilden. **Interpreter** dagegen lesen die Anweisungen einer Sprache und führen zu jeder Anweisung gleich eine Reihe von Maschinenbefehlen aus.

Es gibt auch Zwitter, wie etwa Java, die zunächst in einen **Bytecode** übersetzen und diesen dann interpretieren. Dies hat gegenüber den beiden Extremen Vorteile. Im Unterschied zum Compiler ist das übersetzte Programm vom verwendeten Compiler unabhängig. Die Interpretation von Bytecode ist weitaus schneller als die direkte Interpretation des Programmtextes, allerdings natürlich langsamer als das Ausführen von Maschinensprache.

Grundlegende Aufgabe einer Programmiersprache ist die Reduktion der Komplexität. So wird im einfachsten Fall eine Folge von Maschinenanweisungen in einem Befehl zusammengefasst. Auf der nächsten Stufe der Reduktion steht die Zusammenfassung von Befehlen zu **Unterprogrammen**, die von beliebigen Punkten aus zur Verfügung stehen.

Eine weitere Reduktion der Komplexität bringt die **Objektorientierte Programmierung** (OOP), die Daten und zugehörige Unterprogramme in Objekten zusammenfasst. Java ist eine objektorientierte Sprache. Dazu später mehr. Man könnte die Komplexität noch weiter reduzieren, indem man Objekte zu ganzen Programmteilen zusammenfasst, die vordefinierte Kommunikationsmittel verwenden. Dies wird bei der visuellen Programmierung (z.B. mit Java-Beans oder Eclipse) realisiert.

Zum Entwurf von Programmen braucht man nur die drei Grundstrukturen der Algorithmen, denn nur sie lassen sich in Computerprogramme umsetzen. Zusätzlich benötigt eine Programmiersprache noch folgende Möglichkeiten:

Zuweisungen und Ausnahmeanweisungen

Hochsprachen

Die erste höhere Programmiersprache war der Plankalkül von Konrad Zuse, den er 1942-1946 entwarf, aber nie anwendete (erst 1975 wurde er im Rahmen einer Diplomarbeit implementiert). Etwa um dieselbe Zeit entstand die erste standardisierte Hochsprache **Ada**, benannt nach **Lady Ada Lovelace** (1815–1852), der Tochter von **Lord Byron** und Mitarbeiterin von **Charles Babbage**. Zu den Hochsprachen gehören diejenigen Programmiersprachen, die die Erstellung eines Programms in einer abstrakten Sprache ermöglichen, die zwar für Menschen, aber nicht unmittelbar für Computer verständlich ist.

Familien von Hochsprachen

Algol 60 - Basic – Visual Basic

Fortran - Pascal - Delphi

Ada, Prolog, C

C++, Java (objektorientiert)

esoterische:

Whitespace verwendet das Leerzeichen, das Tab-Zeichen und ASCII255 (minimalistisch)

Brainfuck `+++>+++++<[->+<]` addiere 3

im Detail:

`+++` erhöhe Speicherzelleninhalt drei Mal um 1

`>` gehe zur nächsten Zelle

`+++++` schreibe 5

`<` gehe zu Zelle 1 zurück

`[` Ausführung hinter `]` fortsetzen, wenn Speicherstelle = 0

`-` vermindere um 1

`]` gehe zurück nach `[`

<http://www.bakera.de/dokuwiki/doku.php/schule/prog/br>

James Gosling begann 1990 mit der Entwicklung einer Programmiersprache OAK für Toaster, Videorecorder, etc. 1993 überzeugte er seinen Chef, OAK für das neue www umzustricken, so entstand **Java**.



Die Vorteile sind die Datensicherheit und die Portabilität. Java ist objektorientiert, (im gewissen Sinn) einfach, robust und schnell.

Die Syntax ist ähnlich C und C++ aber es gibt (wegen der Sicherheit) keine pointer, structs, unions, goto und keine Mehrfachvererbung.

Java-Compiler erzeugen keine ausführbaren Programme, sondern plattformunabhängigen, interpretierungsbedürftigen Bytecode in CLASS-Dateien. Der Interpreter ist die Java Virtual Machine (JVM).

Was kann ich mit Java machen? **Alles!**

Animationen, Laufbänder, Chats auf Web-Seiten, Taschenrechner, Spiele, Verschlüsselungsprogramme, Web-Shops, Musik, Textverarbeitungen, Client-Server-Programmpakete, Bildverarbeitungsprogramme, neuartige Benutzeroberflächen, Übersetzungsprogramme, Lernanwendungen, Uhren, Tabellenkalkulationen, Rechtschreibprüfungsprogramme ... Anwendungen für das Android-Betriebssystem werden in Java geschrieben.



Installation der Java-Umgebung

<http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>
<http://www.javaeditor.org/index.php/Download>

Die Bezeichnungen für die verschiedenen Java-Versionen sind verwirrend. Seit der Version 1.1 wird auch von Java2 gesprochen.

Um fertig compilierte Javaprogramme ablaufen zu lassen, benötigt man die 'Java Runtime Environment', wie sie auf fast allen Windowssystemen schon für die Browser installiert ist. Die Erstellung von Programmen erfordert aber das 'Java Development Kit (JDK)', das man bei ['Oracle'](#) herunterladen kann (7u45 ist die aktuelle Version).

Die Unterverzeichnisstruktur von Java ist verbindlich, die *.zip-Dateien werden komprimiert genutzt. Bei Komplikationen bitte überprüfen:

der 'Path' muss ..\BIN enthalten, z.B. C:\programme\java\bin

In Windows 7: Systemsteuerung | System | Erweiterte Systemeinstellungen | Umgebungsvariablen unter Systemvariablen notfalls PATH bearbeiten, falls C:\programme\java\bin nicht drinsteht

Test

Ein harmloses Programm, das keine Voraussetzungen braucht ist **Hallo.java**:

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

Speichere den Code in einer Datei **Hallo.java**, kopiere sie nach C:\programme\java\bin (falls das dein Programm-Ordner ist) und rufe in der cmd-Konsole nacheinander "**javac Hallo.java**" und "**java Hallo**" auf. Jetzt sollte da "Hallo!" stehen - klappt aber nicht bei Windows 7 und 8.

Eine Methode, die Installation zu umgehen, ist folgende (nur bis XP): Kopiere von einer funktionierenden Installation die Ordner C:\programme\java und C:\programme\javaeditor auf einen Stick und kopiere sie auf den eigenen Rechner in C:\programme. Wenn die Umgebungsvariablen deines Rechners dem nicht entgegenstehen, klappt's jetzt auch.

Wir verwenden den **Java-Editor** als Hilfsmittel um Programme zu schreiben, gleich zu compilieren und aufzurufen (mit [F9]). Wenn's nicht klappt, der vorherige Test aber funktioniert, ist es am einfachsten, Du deinstallierst den Java-Editor und installierst ihn neu. Wenn's jetzt immer noch nicht klappt: Lösche alle Java-Installationen aus der Systemsteuerung, lösche alle Ordner Java, JRE oder JDK und installiere erst das JDK, dann den Java-Editor¹ neu. Den Java-Editor gibt es [hier](#).

Der **Java-Editor** füllt sein Konfigurationsmenü bei der Installation selbst aus, manchmal lohnt es sich aber, dort einen Fehler zu suchen:

In Fenster | Konfiguration → Java → Interpreter → JDK-Ordner steht der Ort des Java-Ordners
 Interpreter steht z.B. C:\programme\java\bin\java.exe
 Classpath Admin steht z.B. .;C:\programme\JavaEditor
 → Java → Compiler → Java-Compiler: C:\programme\java\bin\javac.exe
 Parameter: -O -deprecation -g

¹ Der Java-Editor muss nicht installiert werden, der Ordner kann kopiert werden. Dann muss aber die Verknüpfung der Dateieendung von Hand angelegt werden und wahrscheinlich auch die hier beschriebene Konfiguration.

Java-Editor



Datei | *Neu* | *Java* oder [Strg]+[N] öffnet ein leeres Fenster.

Ein Klick auf das erste Monitorsymbol fordert zur Eingabe eines Dateinamen auf und speichert gleich Dateiname.java. Anschließend wird im Textfenster der erforderliche Rahmen für ein neues Programm angelegt. Die Klasse im neuen Programm erhält dabei automatisch den Namen der Datei!

Im Fenster arbeitet man wie mit einem der üblichen Textverarbeitungsprogramme. Markieren, löschen, kopieren, einfügen, ausschneiden, verschieben geht alles mit Maus und den üblichen Tastaturkürzeln. Der Editor unterstützt die Arbeit mit der Windows-Zwischenablage, mehrstufiges Rückgängig machen von Editoroperationen und das Suchen und Ersetzen von Texten.

Sprünge

Wirkung	Taste	Maus
Zeilenanfang	[Pos 1]	L-Klick an die richtige Stelle
Zeilenende	[Ende]	L-Klick an die richtige Stelle
vorige Bildschirmseite	[Bild auf]	L-Klick auf die Bildlaufleiste oberhalb des Rollbalkens
nächste Bildschirmseite	[Bild ab]	L-Klick auf die Bildlaufleiste unterhalb des Rollbalkens
Nächster Wortanfang	[STRG]+[Rechtspfeil]	L-Klick
voriger Wortanfang	[STRG]+[Linkspfeil]	L-Klick
Beginn der vorigen Seite	[STRG]+[Bild auf]	Blauer Doppelpfeil nach oben auf der Bildlaufleiste
Beginn der nächsten Seite	[STRG]+[Bild ab]	Blauer Doppelpfeil nach unten auf der Bildlaufleiste
Bestimmte Zeile finden	[STRG]+[G], Seitenzahl eingeben	Rollbalken verschieben, bis die gesuchte Seitenzahl angezeigt wird
Textanfang	[STRG]+[Pos 1]	Rollbalken ganz rauf ziehen
Textende	[STRG]+[Ende]	Rollbalken ganz runter ziehen

Spezialfunktionen

- ❖ Man kann markierten Texte einrücken (Strg+Umsch+I), ausrücken (Strg+Umsch+U) sowie ein- und auskommentieren (Strg+K).
- ❖ Strg+U fügt System.out.println() in den Quellcode ein,
- ❖ Strg+Y löscht eine Zeile.
- ❖ Bis zu zehn Lesezeichen kann man mit der Maus oder der Tastatur (Strg+Umsch+#) setzen und zu einem Lesezeichen springen (Strg+#).
- ❖ Wählt man neben dem Programm-Kartenreiter den Reiter Kontrollstrukturen, so kann man z.B. die Struktur der if-Verzweigung mit einem Klick eingeben. Auch die Klammern sind schon alle vorhanden.
- ❖ Fenster | Konfiguration | Editor zeigt unter Tastatur eine Liste von Tastenkürzel an.

Javastart

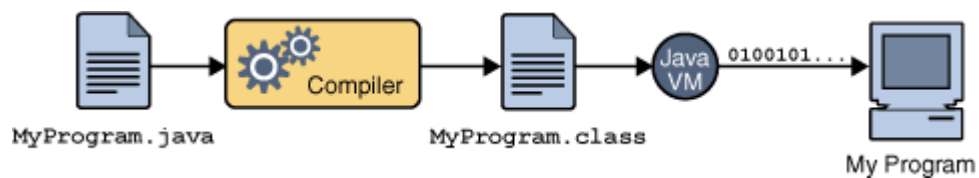
Ein fertiges Programm startet man mit dem grünen Dreieck oder [F9]. Im untersten Feld werden eventuelle Fehler angezeigt. Dort schaut man zuerst nach der angegebenen Stelle: hinter dem Dateinamen steht, jeweils durch Doppelpunkt getrennt, die Zeilennummer und die Position in der Zeile.

Erstellen eines neuen Programms

In der Symbolleiste des JavaEditors unter 'Programm' auf den schwarzen Kasten klicken, das leere Programm benennen (z.B. *Hallo.java*) und in `..\ITG\JavaProgs` speichern. Dadurch ist der erforderliche Java-Rahmen bereits erzeugt und der Dateiname steht auch bereits in der Datei.

Beim Versuch, ein Programm zu compilieren (in ByteCode umzuwandeln) gibt der JavaEditor eventuell Fehlermeldungen am unteren Fensterrand aus. Diese enthalten eine Zeilennummer und eine Spaltennummer – der Fehler kann aber auch davor liegen!

Gelingt die Compilierung des Programms **Hallo.java**, erzeugt Java die Datei **Hallo.class** mit dem Bytecode im selben Ordner. Diese Dateien sind versionsabhängig, mit einer anderen Version erzeugte class-Dateien müssen eventuell gelöscht werden, Java legt sie dann wieder an. Solange wir keine voneinander abhängigen Programmdateien schreiben, können deshalb die class-Dateien jederzeit gelöscht werden.



Übungsaufgaben II

-
1. *Installiere Java JDK daheim auf Deinem PC.*

 2. *Was kann Java nicht*

 3. *Was macht ein Compiler, was macht ein Interpreter?*

 4. *Was bedeutet 'portabel'*

3. Lineare Programme

Aufbau eines einfachen Programms

```
public class ErstesProgramm {
    public static void main(String[] args) {
        String name;           // ein String enthält eine Zeichenkette (gewöhnlich also Text)
        int zahl1,zahl2,zahl3,ergebnis; // int-Variable können nur ganze Zahlen enthalten,
        double zahl4;          // double-Variable enthalten Dezimalzahlen

        System.out.println("Mein erstes Programm");
        name="Egon von J1";
        System.out.println(name);
        zahl1=5;
        zahl2=3;
        ergebnis=zahl1+zahl2;
        System.out.print(zahl1 + " addiert zu " + zahl2 + " ergibt: ");
        System.out.println(ergebnis);
        zahl4=7.0/4.0; System.out.println("7.0/4.0 = " + zahl4);
        zahl4=7/4.0;   System.out.println("7/4.0 = " + zahl4);
        zahl4=7/4;     System.out.println("7/4 = " + zahl4);
        zahl3=7/4;     System.out.println("7/4 = " + zahl3 + " (Die Ergebnisvariable war als int deklariert)");
    }
}
```



```
public class Variable {
    ...
}
```

Der **Kopf**. Vorläufig sind die Details über Klassen für uns nicht relevant. Wichtig ist nur, dass die Java-Datei den gleichen Namen wie die Klasse (hier "Variable") hat.

```
public static void main
(String[] args) {
    ...
}
```

Der **Rumpf**, die Methodenvereinbarung. Hier beginnt (und endet) das (Haupt-) Programm. Er enthält die Vereinbarungen und Befehle und ist selbst wieder in Kopf und Rumpf gegliedert. Die Methode 'main' muss in jedem Programm vorkommen. **Vor dem Methodenname steht der Ereignistyp. Liefert die Methode kein Ergebnis, ist der Typ void. Hinter dem Namen folgt in runden Klammern die Parameterliste, ggf. die leere Liste ().**

```
System.out.println("Erstes
Programm");
```

Ausgabe der Zeichenkette "Erstes Programm" (und Zeilensprung). Die Zeile ist ein Beispiel für eine Anweisung. Jede Anweisung wird mit einem ; (Semikolon) abgeschlossen.

```
// Kommentar
```

Ein **Zeilenkommentar**. Wird vom Compiler ignoriert. Minimum: Autor, Datum, Zweck (Version)

Vorläufig können wir die nebenstehenden Zeilen als Rahmen eines Programms auffassen. Die drei Punkte ... werden durch die relevanten Programmteile ersetzt.

```
public class Programmname {
    public static void main (String[] args) {
        ...
    }
}
```

```
int summand1;
```

Deklaration einer Variablen mit Namen summand1. Variablen, die verwendet werden sollen, müssen erst deklariert werden, damit der Compiler weiß, ob es sich z.B. um eine Dezimalzahl oder ganze Zahl handelt.

Die Variable summand1 ist vom Typ **int** (integer, d.h. ganze Zahl). Sie kann Werte von -2'147'483'648 bis 2'147'483'647 annehmen.

Die allgemeine Form der Variablendeklaration lautet:

```
<datentyp> <variablenname>;
```

In diesem Fall ist der Datentyp int und der Variablenname summand1.

```
summand1 = 5;
```

Dies ist eine Zuweisung. **summand1** wird zu 5.

```
int summand1 = 5;
```

kurz in einer Zeile: **<datentyp> <variablenname>=<anfangswert>**

```
ergebnis = summand1+
summand2;
```

An **ergebnis** wird der Wert zugewiesen, der sich aus der Berechnung des Ausdrucks **summand1+ summand2** ergibt. Dieser Ausdruck ist vom Typ **int**, da '+' hier eine Operation mit zwei **int**-Variablen ist.

Bezeichner

So wie sich manche Leute beim Kopfrechnen Zwischenergebnisse aufschreiben, kann sie auch der PC speichern. Es genügt natürlich nicht, irgendwo in den Hauptspeicher '42' zu speichern, wenn man diese Zahl nicht wiederfinden kann (ohne sie zu kennen). Daher muss eine solche Zahl erst einen Namen bekommen, und diesem Namen wird ein bestimmter Bereich im Speicher zugewiesen.

Namen in Computersprachen nennt man 'Bezeichner'. Gleichgültig ob für eine Variable, eine Konstante, eine Klasse oder ... Bezeichner müssen eindeutig sein. Sie beginnen mit einem Buchstaben, **einem \$-Zeichen oder einem Unterstrich (beides nicht üblich)**, dann dürfen Ziffern, Buchstaben oder der Unterstrich (unbeliebt) folgen. Java unterscheidet Groß- und Kleinschreibung aufs Genaueste. Bevor die bezeichneten Begriffe verwendet werden können, muss Java Speicherplatz für sie reservieren. Seine Größe hängt vom Typ ab, deshalb steht die **Deklaration** am Anfang eines Programms. Vor der erstmaligen Verwendung, muss einer Variablen ein Wert zugewiesen werden (**Initialisierung**) anderenfalls wird der Wert Null verwendet, in besonderen Fällen wird der Compiler die fehlende Initialisierung monieren.

Pascal	Java	Python
<pre>VAR m,n : Integer; Antwort: Boolean; x : Real; Zeichen: Char; BEGIN m := 5; n := 1; Antwort := FALSE; x := 3.14159; Zeichen := "G"; END.</pre>	<pre>int m,n; boolean Antwort; float x; char Zeichen; m = 5; n = 1; Antwort = false; x = 3.14159; Zeichen = "G";</pre>	<pre>m = 5 n = 1 Antwort = 0 x = 3.14159 Zeichen = "G"</pre>

Die Tabelle zeigt, wie Variablen in verschiedenen Programmiersprachen deklariert und initialisiert werden. Finde die Unterschiede!

Schreibweisen

Java reagiert empfindlich auf Verstöße gegen formale Vorschriften.

- ✦ **Klassennamen** (hier die Programmnamen) werden groß geschrieben, gerne auch mit Großbuchstaben im Inneren des Wortes (Kamelschreibweise): *'TollesBeispiel'*.
- ✦ Die **Datei** heißt wie die Klasse (mit *'.java'* dahinter). Auch in Windows kann man Dateinamen groß oder klein schreiben, auch wenn das Betriebssystem den Unterschied scheinbar ignoriert! Der Java-Editor richtet's normalerweise alleine.
- ✦ **Variablen** werden klein geschrieben. Man verwendet gerne aussagekräftige Namen, also *'steuer'* statt *'s'*. 'Innere' Wörter beginnen mit einem Großbuchstaben: *anzahlDerFragen*
- ✦ Keine unnötigen Variablen deklarieren, die man gar nicht braucht!
- ✦ **Bezeichner**, also auch Klassen- oder Variablennamen, enthalten keine Satz- oder Sonderzeichen – auch kein Leerzeichen – höchstens mal (ist aber verpönt) einen Unterstrich *'Tolles_Beispiel'*.
- ✦ Reservierte Wörter, wie *double*, *class*, ... dürfen (können) nicht als Bezeichner verwendet werden.
- ✦ **Konstanten** schreibt man ausschließlich mit Großbuchstaben: *MEHRWERTSTEUERSATZ=0.19*
- ✦ Das **Dezimaltrennzeichen** ist ein Punkt.
- ✦ Soll ein Objekt aus einer Klasse aufgerufen werden, setzt man zwischen Klassennamen und Objektname einen **Punkt**: *x=Math.sin(2);*
- ✦ Zusammengehörende Befehlsblöcke werden in **{ }** eingeschlossen.
- ✦ **{ }**-Klammern treten immer **paarweise** auf! '{' steht entweder hinter der Anweisung – dann steht '}' in derselben Spalte wie das erste Zeichen der Anweisung – oder (besser) '{' und '}' stehen in je einer eigenen Zeile in der selben Spalte.
- ✦ Enthält ein Anweisungsblock nur eine einzige Anweisung, können die geschweiften Klammern entfallen.
- ✦ Wenn die **{ }**-Klammern leer sind, heißt das, dieser Block enthält keine Anweisung.
- ✦ Jede Anweisung endet mit einem **Semikolon**. Meist steht in einer Zeile nur eine Anweisung, es können aber auch mehrere sein.
- ✦ **//** leitet einen kurzen Kommentar ein, auch in derselben Zeile hinter einem Befehl.
- ✦ Lange Kommentare stehen zwischen **/*** und ***/**
- ✦ **Leerzeilen** und **Kommentare** im Programm wirken sich nicht aus. Kommentare dienen der Dokumentation und erhöhen die Lesbarkeit und Verständlichkeit eines Programms.

Zahlendatentypen

Alle Objekte, die Variablen zugewiesen werden können, nennt man Daten. Handelt es sich dabei um Zahlen, ist für die Initialisierung wichtig zu wissen, wie viel Platz im Arbeitsspeicher verwendet werden soll. Generell nimmt man immer den Typ, der den Anforderungen genügt und den geringsten Platzverbrauch hat.

Typ	Beschreibung	Wertebereich
byte	8-Bit-Ganzzahl Zweierkomplement	-128 ... 0 ... 127 ($=2^8-1$)
short	16-Bit-Ganzzahl	-32 768 ... 0 ... 32 767
int	32-Bit-Ganzzahl (0x3FF (hexagesimal), 0377 (octal))	-2 147 483 648 ... 0 ... 2 147 483 647
long	64-Bit-Ganzzahl	-9,223,372,036,854,775,808 ... 9,223,372,036,854,775,807 ($-2^{63} \dots 2^{63}-1$)
float	32-Bit-Gleitkommazahl	ca. $\pm 3.40282347 \cdot 10^{38}$; kleinste Zahl > 0: $1.4 \cdot 10^{-45}$
double	64-Bit-Gleitkommazahl (-1E10)	ca. $\pm 1.79769313486231570 \cdot 10^{308}$ $4.9 \cdot 10^{-324}$ ist etwa die kleinste Zahl über Null

Für uns sind vorerst die Typen double und int ausreichend.

Ist in einem Ausdruck eine double-Zahl und eine int-Zahl enthalten, so ist das Ergebnis vom Typ double.

Achtung!

Der Ausdruck $1/3$ ergibt 0, da er ganzzahlig berechnet wird (1 und 3 sind int-Zahlen)

Will man $0.333\dots$ als Ergebnis erhalten, so kann man $1.0/3$ schreiben. In diesem Fall ist der erste Operand double, und daher auch das Ergebnis. Natürlich geht auch $1.0/3.0$ oder $1/3.0$.

Nun können wir etwa einen komplizierteren Ausdruck berechnen.

```
public class Test {
    public static void main (String[] args) {
        double x=1.5,y;
        y=1+x*x*x/2+x*x*x*x/6+x*x*x*x*x/24;
        System.out.println("y = "+y);
    }
}
```

Es ist also möglich, mehrere Variable in einer Zeile zu deklarieren und ihnen auch sofort Werte zuzuweisen. Man beachte die Addition eines Textes ("y = ") und einer double-Variablen bei der Ausgabe. Dafür wird intern erst der double-Wert in einen Text umgewandelt und danach werden beide Texte aneinander gehängt. Das Programm erstellt folgende Ausgabe

y = 4.3984375

Eingaben

Java sieht standardmäßig keine Eingaben von der Tastatur vor. Wir verwenden dafür ein Unterprogramm Console.class, das im selben Ordner wie die *.java-Datei liegen soll.

<code>j = Console.readInt();</code>	integer-Wert wird eingelesen und der Variablen j zugeordnet.
<code>k = Console.readInt("Gib k ein! ");</code>	Auf dem Monitor erscheint die Aufforderung "Gib k ein! " Die eingegebene Zahl wird k zugeordnet.
<code>m = Console.readDouble()</code>	analog für Dezimalzahlen
<code>n = Console.readDouble("Eingabe: ")</code>	analog für Dezimalzahlen

Arithmetische Operatoren

Für die arithmetischen Datentypen (das sind alle außer boolean) sind die fünf Operatoren `+`, `-`, `*`, `/` und `%` (modulo, d.h. Restberechnung bei Division) definiert. Das Ergebnis ist jeweils vom gleichen Typ. Deswegen wird bei der Division zweier ganzer Zahlen nur der ganzzahlige Anteil berechnet.

```
int n=3;
System.out.println(n/4); // druckt 0
```

Die Ausführungsreihenfolge von Operatoren ist streng festgelegt, entspricht aber der üblichen Konvention ("Punkt vor Strich", Berücksichtigung von Klammern, sonst von links nach rechts).

```
System.out.println((3.0+7.0)/(4.0*5.0+80.0)); // druckt 0.1
```

Der Operator `^` ("hoch") ist nicht definiert. Hier muss die mathematische Funktion **Math.pow** verwendet werden (**Math.pow(Basis, Exponent)**).

Es gibt auch die (Postfix-) Operatoren `++` und `--`. Diese Operatoren erhöhen, bzw. erniedrigen, einen ganzzahligen Wert um 1.

```
Beispiel:      int i=1;
                i++;
                System.out.println(i); // druckt 2
```

Zuweisungen

Kurzform	Beispiel		Langform
<code>=</code>	<code>n=9</code>	einfache Zuweisung	
<code>+= (-=)</code>	<code>n+=9 (n-=9)</code>	Die Zunahme (Abnahme) von n beträgt 9.	<code>n=n+9 (n=n-9)</code>
<code>++ (--)</code>	<code>i++ (i--)</code>	Inkrement (Dekrement)	<code>i=i+1 (i=i-1)</code>
<code>*=</code>	<code>n*=9</code>	multipliziere n mit 9	<code>n=n*9</code>
<code>/=</code>	<code>n/=9</code>	dividiere n durch 4	<code>n=n/4</code>
<code>%=</code>	<code>n%=10</code>	modulo, hier 'Einerziffer'	<code>n=n%10</code>

prefix +i ?

Mathematische Funktionen

Die mathematischen Funktionen sind in der Klasse **Math** erklärt. Das bedeutet, vor ihrem eigentlichen Namen steht **Math** mit einem Punkt. Sie werden mit `import java.util.*` vor dem eigentlichen Programm eingebunden und wie im folgenden Beispiel aufgerufen:

```
double x;
x=Math.sin(2);
System.out.println("Der Sinus von 2 ist " + x );
```

Es gibt die üblichen Funktionen **abs**, **sin**, **cos**, **exp**, **log**, **sqrt**, **asin** und **acos**. Daneben existieren Funktionen mit zwei Variablen, wie **pow** (berechnet x^y), **max** und **min**. Alle diese Funktionen müssen mit vorangestelltem **Math.** aufgerufen werden.

Außerdem existieren die Funktionen **floor** und **ceil**, die zur nächsten ganzen Zahl ab-, bzw. aufrunden. Die Funktion **round** rundet zur nächsten ganzen Zahl, ergibt aber einen long-Wert.

Schließlich erzeugt die Funktion **random** eine Zufallszahl zwischen 0 und 1.

Andere Datentypen

In der Tabelle sind jeweils der Java-Name des Datentyps, Beispiele für Konstanten des Datentyps und die möglichen Werte wiedergegeben, die Variablen dieses Datentyps annehmen können.

Datentyp	Bsp. für Konstanten	Wertebereich
char	'A', 'a', ' ', '\t', '\n', '\xFE', '\u0121' (Unicode)	Zwei byte-Character in Unicode. Wenn das oberste Byte 0 ist, so entspricht dies dem normalen Zeichensatz. In anderen Sprachen ist dieser Datentyp nur ein Byte lang. '\u0000' (=0) ... '\uffff' (=65535)
boolean	true, false	WAHR oder FALSCH

Ausgaben

Verwendung

<code>System.out.print(Elemente);</code>	<i>Elemente</i> sind Variablen oder Zeichenketten (z.B. Texte) in doppelten Anführungszeichen)
<code>System.out.print("Text");</code>	schreibt das Wort 'Text' auf den Bildschirm, der Cursor steht dann hinter 't'
<code>System.out.println(Elemente);</code>	-ln bewirkt einen Zeilenumbruch und setzt den Cursor an den Beginn der nächsten Zeile
<code>System.out.println();</code>	gibt eine Leerzeile aus
<code>System.out.print("\n\n\n");</code>	macht drei Leerzeilen
<code>System.out.print("2,95 " + "Euro");</code>	fügt Zeichenketten (Strings) aneinander. Beachte das Leerzeichen hinter '95'!

Methoden zum freien Positionieren des Textes gibt es nicht!

Übungsaufgaben III

-
1. *Verändere das erste Beispielprogramm und beobachte, was passiert:*
 - a) 1. Zeile **Hallo** statt **ErstesProgramm**
 - b) 1. Zeile **erstesProgramm** statt **ErstesProgramm**
 - c) 3. Zeile **Public** statt **public**
 - d) Lösche ein Semikolon mitten im Programm!

 2. *Ergänze das Programm "ErstesProgramm" so, dass außerdem die Differenz, das Produkt und der Quotient der Zahlen berechnet und ausgegeben wird. Was fällt auf?*

 3. *Füge folgende Zeile nach `ergebnis=zahl1+zahl2` ein: `ergebnis++`; Was bewirkt das?*

 4. *Löse Aufgabe 3 mit `ergebnis--`;*

 5. *Produziere eine Virenmeldung als Bildschirmausgabe:*

Virenangriff!
Von CD booten und
Virensoftware starten!

 6. *Lass den PC deine Visitenkarte schreiben!*

 7. *Erstelle einen vollständigen Kassenbon für den Einkauf einer CD zum Nettopreis von 17,20€, zuzüglich 19% Mehrwertsteuer bei der Firma CD-Schrott in Graceland.*

 8. *Schreibe ein Programm, welches eine Variable "**kantenLaenge**" enthält und das Volumen sowie die Oberfläche eines Würfels bei eingegebener Kantenlänge berechnet und ausgibt. Erstelle hierzu ein Projekt "**Wuerfel**".*

 9. *Aus Grundradius und Höhe eines Kegels sollen Oberfläche und Volumen berechnet werden.*

 10. *Schreibe ein Programm, das den Umfang eines Kreises mit Radius 2.0 ausgibt. Die Ausgabe sollte so aussehen:*
Der Umfang eines Kreises mit Radius 2.0 ist 12.566370614359172
*Verwende die mathematische Konstante **Math.PI**.*

 11. *Was ist das Ergebnis des Ausdrucks $1000000*1000000$? Warum entsteht dieses Ergebnis?*

 12. *Was ergibt $0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1-1.0$? Warum ist das nicht exakt 0.0?*

 13. *Ändere ein Programm so ab, dass die Zahlen von der Tastatur eingegeben werden können. Dazu musst Du **Console.class** in Deinen Java-Ordner kopieren.*
"Console.class" muss in dem Ordner liegen, in dem die aufgerufene Java-Datei liegt!

 14. *Schreibe ein Programm "Durchschnitt", das den Durchschnitt von vier eingegebenen Zahlen berechnet.*

-
15. Schreibe ein Programm, das nach Eingabe zweier Zählerstände die Jahresrechnung des Stromlieferanten erstellt. Verwende eine echte Vorlage, um alle Bestandteile des Preises erfassen zu können.
-
16. Schreibe ein Programm, welches die Wurzel einer eingegebenen Zahl berechnet und ausgibt.
-
17. Nach der Eingabe zweier Widerstandswerte soll der Ersatzwiderstand bei Reihen- und Parallelschaltung ausgegeben werden.
-
18. Für einen Kredit bekannter Höhe mit bekanntem Jahreszins und gegebener Monatsrate, soll die Kredithöhe nach einem Monat berechnet werden.
-
19. Ein Programm soll die Werte zweier Variablen vertauschen. Z.B. war laut Festlegung $a=5$ und $b=10$, aber die Variable a soll anschließend den Wert 10, und b den Wert 5 haben.
-
20. Ein Programm soll die Quersumme einer zweistelligen (dreistelligen) Zahl berechnen.
-
21. Man kann mit dem Datentyp **char** rechnen. Er wird dann wie **int** behandelt, indem sein Code in eine Zahl umgewandelt wird. Berechne den 10. Buchstaben nach dem 'A'! Wie viele Buchstaben liegen zwischen 'A' und 'Z'? Wo liegen die kleinen, und wo die großen Buchstaben, wo die Zahlen?
-
22. Teste die Verschiebungs-Operatoren $>>$, $>>>$ und $<<$. Was bewirken diese Operatoren und warum?
-
23. Schreibe ein Programm, welches aus Tag, Monat und Jahr den Wochentag berechnet. (Die Problematik mit dem Januar und Februar muss nicht beachtet werden.)
Hierzu wird folgende Formel verwendet:

$$z = \left(t + \frac{(m+1)26}{10} + \frac{5j}{4} + \frac{H}{4} - 2H - 1 \right) : 7$$

(bei allen Divisionen ist nur der ganzzahlige Teil zu nehmen)

Es ist: t : Tag, m : Monat, H : Hunderterzahl des Jahres, j zweistelliges Jahr in diesem Hunderter
z.B. 25.5.1990: $t=25$, $m=5$, $H=19$, $j=90$

Von der Division durch 7 interessiert nur der Rest, der den Wochentag angibt: 0 = Sonntag, 1 = Montag, 2=Dienstag

Die Monatsnummern 1 und 2 werden nicht verwendet. Handelt es sich um Januar als Monat, so muss der 13. Monat des Vorjahres verwendet werden, entsprechend auch beim Februar der 14. z.B. 12.1.2014: $t=12$, $m=13$, $H=20$, $j=14$. Welchen historischen Grund hat das?

4. Verzweigungen

4.1 if-Verzweigung

Bisher haben wir nur Programme betrachtet, bei denen alle Anweisungen nacheinander ausgeführt werden. Um auf den Ablauf von Programmen Einfluss nehmen zu können, müssen Programmiersprachen über **Kontrollstrukturen** verfügen. Je nach aktueller Bedingung muss es möglich sein, unterschiedliche Wege zu wählen, also eine Wahl treffen zu können. Solche Verzweigungen im Programmablauf werden in Java mit folgender Syntax erreicht:

```

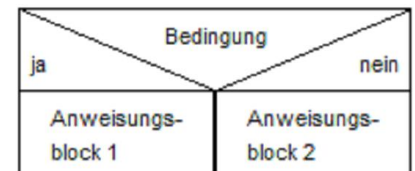
if (<boolescher Ausdruck>)
{
    <Anweisung 1>;
    <Anweisung 2>;
    ...
    <Anweisung n>;
}
else
{
    <Anweisung 1>;
    <Anweisung 2>;
    ...
    <Anweisung m>;
}

```

Unter einem *booleschen Ausdruck* versteht man einen Ausdruck der die Werte true (wahr) oder false (falsch) annehmen kann

Geschweifte Klammern nicht vergessen - sonst gehört nur die erste Anweisung zur Auswahl, die anderen werden als Sequenz ausgeführt.

Im true-Fall werden die Anweisungen des "if-Blocks" ausgeführt. Hat der boolesche Ausdruck dagegen den Wert false, werden die Anweisungen des else-Blocks abgearbeitet.



Struktogramm

```

if (x<0)           // Die Bedingung muss in runden Klammern stehen
{
    System.out.println("x war negativ"
    x=0;
}
else              //wird nur ausgeführt, wenn die Bedingung (...) nicht erfüllt ist
{
    System.out.println("x war positiv oder Null"
}

```

einseitige Verzweigung:

Die **else**-Anweisung kann entfallen: **if (bedingung) {anweisungen}** führt die Anweisungen aus, falls die Bedingung zutrifft. Anderenfalls wird dahinter fortgefahren.

Vergleiche

Weil das Gleichheitszeichen für den Zuweisungsoperator verwendet wird, müssen für den Vergleichsoperator zwei Gleichheitszeichen geschrieben werden: `if a==b`

Operator	>	<	>=	<=	==	!=
Bedeutung	größer als	kleiner als	größer als oder gleich	kleiner als oder gleich	gleich	ungleich

Aussagen werden mit `&&` (UND) und `||` (ODER) verknüpft. Sie werden mit `!` (nicht) ins Gegenteil verkehrt.

```
if ((x<2)|| (x>4)) ... //wenn x kleiner 2 oder größer 4 ist, dann ...
```

Bemerkungen

Falls mehrere `if`-Abfragen geschachtelt werden, so bezieht sich das `else` jeweils auf das letzte vorangehende `if`, natürlich unabhängig von der Einrückung.

```
if (x<0)
    if (x<-2)
        System.out.println("x ist kleiner als -2.");
    else System.out.println("x liegt zwischen -2 und 0");
else System.out.println("x ist positiv");
```

Soll das `else` zu einem weiter davor stehenden `if` gehören (weil die innere Verzweigung keinen `else`-Teil hat), wird ein `else` mit der leeren Anweisung `{}` eingefügt.

```
if (bedingung1)
    if (bedingung2) {anweisungen}
    else {}
else {anweisungen}
```

Fehler

Der Java-Compiler kann nur **Syntax-Fehler** entdecken, das sind Verstöße gegen die Regeln der Programmiersprache.

Semantische Fehler sind logische Probleme, z.B. wenn man einen Fall vergisst, oder versäumt, zusammen gehörende Anweisungen nach `if` in einen Anweisungsblock `{}` zu setzen. Das Programm macht also nicht genau das, was der Programmierer beabsichtigt hat.

Das ist vermeidbar durch:

- ❖ sorgfältiges Strukturieren von Anfang an
- ❖ bereits vorhandene, funktionierende Programmstücke wieder verwenden
- ❖ ausgiebig testen

Beispiele

Anreden

Pascal	Java	Python
<pre>if Geschlecht = 'w' then { Anrede := 'Frau'; } else { Anrede := 'Herr'; }</pre>	<pre>if (Geschlecht == "w") { Anrede = "Frau"; } else { Anrede = "Herr"; }</pre>	<pre>if Geschlecht == "w": Anrede = "Frau" else: Anrede = "Herr"</pre>

Python verzichtet auf Klammern, dafür müssen die Anweisungen eingerückt werden!

Uhr

(unter Verwendung der Systemzeit)

Um später nicht umlernen zu müssen, werden wir von Anfang an die richtigen Bezeichnungen (z.B. Objekt, Klasse und Methoden) benutzen, auch wenn diese Begriffe erst im Laufe des Kurses sauber eingeführt werden.

Es soll die aktuelle Uhrzeit ermittelt werden, und anschließend in Abhängigkeit von der Uhrzeit

"Guten Morgen! ..." bzw.

"Guten Tag! Es ist jetzt ... Uhr." ausgegeben werden.

Wir verwenden die vordefinierte Klasse `Date`. Hierbei wird die Systemuhr des Rechners nach dem Datum gefragt. Dabei enthält das Datum auch Informationen über die Uhrzeit, wie aktuelle Stunde, Minute, etc.)

```
import java.util.*;

public class Uhr {
    public static void main(String[] args) {
        Date datum = new Date();
        int h = datum.getHours();
        int min = datum.getMinutes();
        if (h < 12) {
            System.out.print("Guten Morgen! ");
        }
        else {
            System.out.print("Guten Tag! ");
        }
        System.out.println("Es ist jetzt: " + h + ":" + min + " Uhr.");
    }
}
```

In der Klasse `Date` kann man die oben genannten Informationen bekommen. Dazu wird ein Objekt `datum` der Klasse erzeugt:

```
Date datum = new Date();
```

Mit den Methoden `getHour()` und `getMinutes()`, die man auf das Objekt `datum` anwendet, werden die Stunde und die Minute ermittelt und der Wert in einer `int`-Variablen `h` bzw. `min` gespeichert. Die Klasse `Date` liegt im Paket `java.util`, weshalb wir dieses Paket mit `import java.util.*`; bereitstellen müssen.

4.2 Mehrfachverzweigung (switch)

Es kann auch vorkommen, dass man in einem Programmablauf an eine Stelle kommt, an der mehr als zwei Möglichkeiten angeboten werden sollen, z.B. in einem Menü. Dann muss man, nach unseren jetzigen Kenntnissen, die Mehrfachverzweigung mit einer mehr oder weniger komplizierten Schachtelung von **if**-Anweisungen realisieren. Eine if-Schachtelung für 4 Fälle hat dann z.B. folgende Gestalt:

```

.....
if(Bedingung 1){
    Anweisungen für Fall 1;
}
else {
    if (Bedingung 2){
        Anweisungen für Fall 2;
    }
    else {
        if (Bedingung 3){
            Anweisungen für Fall 3;
        }
        else {
            Anweisungen für Fall 4;
        }
    }
}
}
.....

```

Man kann auf die else-Anweisungen verzichten, um das Programm übersichtlicher zu gestalten. Die gewonnene Übersichtlichkeit erkaufte man sich allerdings mit einem Nachteil. Er besteht darin, dass dann jede if-Anweisung ausgewertet werden muss, gleichgültig, ob eine vorausgegangene if-Anweisung schon erfolgreich war.

Java bietet für diesen Fall, wie auch andere Programmiersprachen eine mehrfache Verzweigungsstruktur an, die einen sogenannten Selektor verwendet. Dieser muss von einfachem Datentyp sein, wie **char**, **byte**, **short** oder **int**. Das reservierte Wort **switch** leitet die Auswahl ein und die **case**-Anweisungen unterscheiden die verschiedenen Fälle.

Wert(ebereich) 1	Wert(ebereich) 2	Wert(ebereich) 3	Wert(ebereich) n	Variable sonst
Anweisungs- block 1	Anweisungs- block 2	Anweisungs- block 3	Anweisungs- block n	Alternativ- block (optional)

Das Struktogramm für eine Mehrfachverzweigung

Diese Struktur ist besonders bei mehr als drei abzu prüfenden Bedingungen vorteilhaft. Der Wert von **"Variable"** kann bedingt auf Gleichheit wie auch auf Bereiche (größer/kleiner bei Zahlen) geprüft werden und der entsprechend zutreffende "Fall" mit dem zugehörigen Anweisungsblock wird durchlaufen. Eine Fallauswahl kann immer in eine Mehrfachauswahl umgewandelt werden.

```

...
int selektor;
selektor = .....//Initialisierung des Selektors, z.B. Eingabe eines Menüpunktes

switch (selektor) {
    case 1 : Anweisung1;    break;
    case 2 : Anweisung2;    break;
    case 3 : Anweisung3;    break;
    default : Anweisung_sonst;
}
// Ende von switch

```

Hat der Selektor z.B. den Wert 2, so werden die Anweisungen des 2. Falles ausgeführt. Die Rolle des "break", wollen wir an folgendem Programm kennen zu lernen, das zunächst ohne "break" ausgeführt werden soll.

Noten

Es soll zu einem als Zahl (1 bis 6) eingegebenen Notenwert der zugehörige Text "sehr gut", "gut", ... ausgegeben werden.

```
{
    System.out.print("Bitte Note eingeben (1..6): ");
    int zahlZensur = Console.readInt();
    switch (zahlZensur) {
        case 1: System.out.println("sehr gut");
        case 2: System.out.println("gut");
        .....
        case 6: System.out.println("ungenuegend");
        default: System.out.println("Falsche Eingabe!");
    }
}
```

Java liefert bei der Eingabe von 3 die nachfolgenden Ausgaben:

```
Bitte Note eingeben (1..6): 3
befriedigend
ausreichend
mangelhaft
ungenuegend
Falsche Eingabe!
```

Die **case**-Nummern sind nämlich lediglich Marken (**Labels**). Hat der Selektor etwa den Wert 2, so springt das Programm an die Marke **case 2:** und führt alle(!) folgenden Anweisungen im **switch**-Block aus. Die mit **case** bezeichneten Stellen im Programm sind also keine Verzweiger sondern lediglich Einstiegsmarkierer. Damit ist aber die gewünschte Funktionalität einer Mehrfachverzweigung noch nicht erreicht. Diese schafft man erst, wenn man den **switch**-Block durch einen Sprung wieder verlässt. Mit **break** realisiert man genau solche Sprünge:

```
{
    System.out.print("Bitte Note eingeben (1..6): ");
    int zahlZensur = Console.readInt();
    switch (zahlZensur) {
        case 1: System.out.println("sehr gut"); break;
        case 2: System.out.println("gut"); break;
        .....
        case 6: System.out.println("ungenuegend"); break;
        default: System.out.println("Falsche Eingabe!");
    }
}
```

Java liefert nun bei der Eingabe von 3 die gewünschte Ausgabe.

Hinweis

Nicht jede verschachtelte **if**-Anweisung lässt sich in eine **switch**-Anweisung überführen. Während bei der **if**-Anweisung boolesche Ausdrücke ausgewertet werden, verzweigt **switch** an Hand der Belegung des Selektors, der wie oben angegeben von einfachem Datentyp sein muss. So gesehen ist die **if**-Anweisung universeller einsetzbar.

Übungsaufgaben IV

-
1. *Wahlweise Umrechnung von SFr in Euro und umgekehrt (verwende den aktuellen Kurs!).*

 2. *Nach Eingabe von Körpergröße und Gewicht wird der Body-Mass-Index berechnet und eine entsprechende Bewertung ausgegeben.*

 3. *Nach Eingabe eines Briefgewichtes wird das Porto ausgegeben.*

 4. *Ein Programm soll die Quersumme einer zweistelligen (dreistelligen) Zahl berechnen und mitteilen, ob die Zahl durch 3 teilbar ist.*

 5. *Ein Programm soll mitteilen, ob die eingegebene vielstellige Zahl durch 187 teilbar ist.*

 6. *Überprüfe, ob eine einzugebende natürliche Zahl eine Quadratzahl ist.*

 7. *Berechne die Lösungsmenge einer quadratischen Gleichung.*

 8. *Errate eine vom PC gewählte Zufallszahl. `zufallszahl=Math.random()` erzeugt eine Zufallszahl zwischen 0 und 1.*

 9. *Berechne ein Dreieck: Eingegeben werden drei Seitenlängen, das Programm gibt Fläche, Umfang und Höhen des Dreiecks aus, sowie die Angabe ob das Dreieck rechtwinklig, gleichschenkelig oder gleichseitig ist.*

 10. *Berechne den Schnittpunkt zweier Geraden.*

 11. *Löse lineare Gleichungssysteme.*

 12. *Programmiere einen Software-Taschenrechner, der einzugebende Zahlen wahlweise addiert, subtrahiert, multipliziert oder dividiert.*

5. Wiederholungen

5.1 Zählschleifen

5.1.1 Einfache Zählschleifen

Folgendes Programm zählt bis 10 und endet dann:

```
for (int i=1; i<=10; i++) // oder ausführlicher: i=i+1;
{System.out.println(i);}
```

Allgemein lautet die Anweisung:

```
for (Anfangsanweisung; Abbruchbedingung; Schleifenanweisung)
{ Anweisungen }
```

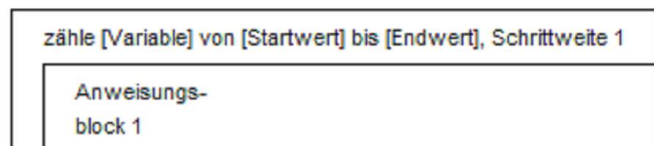
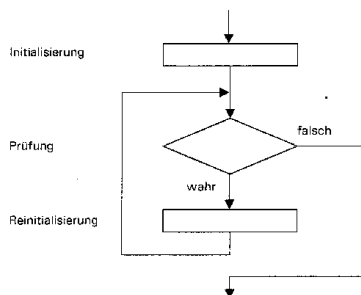
Schleifenkopf	<i>dahinter steht kein Semikolon!</i>
Anfangsanweisung	wird zu Beginn des ersten Schleifendurchlaufs ausgeführt, im Beispiel wird i zu 1.
Abbruchbedingung	Dieser Ausdruck wird vor jedem Durchlauf – auch vor dem ersten – ausgewertet. Fällt die Auswertung positiv aus, so wird ein neuer Durchlauf gestartet, anderenfalls wird die Schleife beendet. Im obigen Beispiel endet das Programm, wenn i größer als 10 wird, d.h. sie wird ausgeführt, solange i<=10 ist.
Schleifenanweisung	Diese Anweisung wird am Ende eines jeden Schleifendurchlaufs ausgeführt. Im Beispiel wird i um 1 erhöht.
Schleifenrumpf	
{Anweisungen}	Der Schleifenrumpf enthält die Anweisungen, die in einem Schleifendurchlauf ausgeführt werden.

Das folgende Beispiel zählt rückwärts von 10 bis 1: `for (int i=10; i>=1; i--)`

Auch hier wurde die Variable wieder in der Schleife deklariert.

Bemerkungen

Die Anfangsanweisung (Initialisierung) kann leer sein, oder auch mehrere, durch Kommata getrennte Ausdrücke enthalten.



Struktogramm einer zählergesteuerten Schleife

Flussdiagramm der for-Schleife

Wichtig!

Die folgende Schleife ist nicht korrekt:

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

Das Semikolon vor dem Schleifenblock ist falsch. Es wird als Leerbefehl interpretiert, der dann 10-mal ausgeführt wird. Danach wird der Schleifenblock einmal ausgeführt mit dem Wert 10 für i.

Beispiele

Fibonacci-Folge

auf Papier auswerten lassen! a|b|i|c

```
int a=1, b=1;
for (int i=3; i<=10; i++)
{
    int c=a+b;
    System.out.println(c);
    a=b; b=c;
}
```

Tabelle von Funktionswerten 1

```
double x,y;
for (x=0 ; x<=8; x+=.5)
{
    y = Math.sin(x)/(Math.sqrt(x*x+1));
    System.out.println(x+" "+y );
}
```

Zufallszahlen

Es werden 100 Zufallszahlen erzeugt und die größte davon ausgegeben.

Math.random() erzeugt eine Zufallszahl zwischen 0 und 1.

Sinnvolles Vorgehen:

- ❖ Erzeuge die erste Zahl und merke sie als (vorläufiges) Maximum.
- ❖ Erzeuge die restlichen Zufallszahlen und vergleiche jede nacheinander mit dem Maximum. Falls die neue Zahl größer ist, wird das Maximum aktualisiert.

```
int i;
double x, max;

max = Math.random(); // erste Zufallszahl, also vorläufig die größte
for (i=2; i<=100; i++) {
    x = Math.random(); // nächste Zahl
    if (x > max)
        max = x; // Aktualisierung des Maximums
}
System.out.println("Maximum: "+max);
```


5.1.2 Schachtelung von Schleifen

Einmaleins

```
int i,j,p;
for (i=1; i<10; i++)
{
    for (j=1; j<10; j++)
    {
        p=i*j;
        System.out.print(p + " "); // gibt alles in einer Zeile aus
    }
    System.out.println(""); // neue Zeile
}
```

Zur Platzersparnis werden hier in der inneren Schleife ganze Zeilen ausgegeben, der Zeilenumbruch kommt später. Die Formatierung lässt sich verbessern, indem die Zahlen untereinander ausgegeben werden:

```
int i,j,p;
for (i=1; i<10; i++)
{
    for (j=1; j<10; j++)
    {
        p=i*j;
        if (p<10)
        {
            System.out.print(" "+p+" "); // einst. Zahlen mit zus. Leerz.
        }
        else
        {
            System.out.print(p+" "); // zweistellige Zahlen
        }
    }
    System.out.println("");
}
```

ASCII-Zeichensatz

Wir wollen den ASCII-Zeichensatz ausgeben. Dies sind die char-Werte von 32 bis 127 (die Werte von 0 bis 31 sind Sonderzeichen, z.B. zur Druckersteuerung).

```
{ int i,j;
  for (i=2; i<16; i++) // 14 Zeilen
  { for (j=0; j<16; j++) System.out.print((char)(16*i+j));
    // Jeweils 16 Zeichen pro Zeile
    System.out.println(" "+i*16+" - "+(i*16+15));
    // Am Zeilenende der Bereich der Zeichen in dieser Zeile.
  }
}
```

Die Ausgabe ist

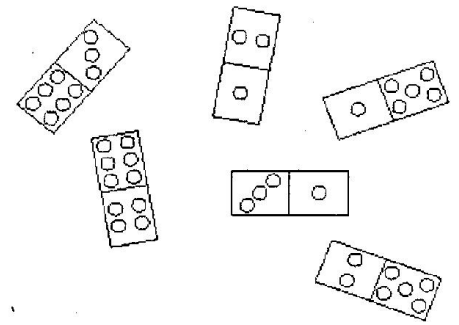
```
!"#$%&'()*+,-./ 32 - 47
0123456789:;<=>? 48 - 63
@ABCDEFGHIJKLMNO 64 - 79
PQRSTUVWXYZ[\]^_ 80 - 95
`abcdefghijklmno 96 - 111
pqrstuvwxyz{|}~ 112 - 127
...
```

5.1.3 Domino

Beispiel vom Nov.02

Dieser Einschub führt nochmals schrittweise zur Wiederholungsstruktur.

Auf dem Tisch liegen 6 Dominosteine. Diese sollen in eine Längsreihe (Reihenfolge beliebig) gelegt werden.



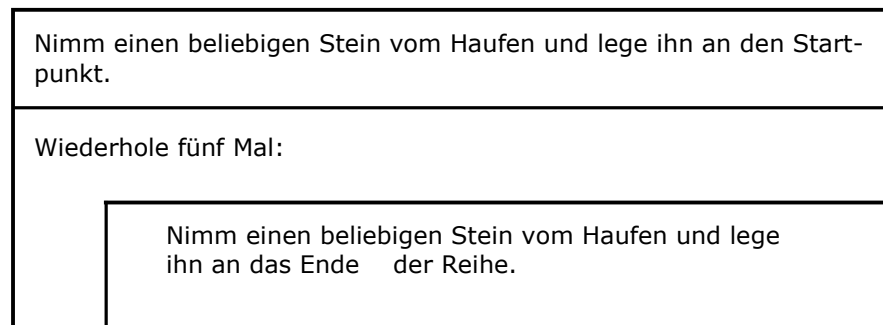
Algorithmus (Sequenz):

Beachte: Das Ergebnis nach Ablauf des Algorithmus muss nicht immer gleich sein!

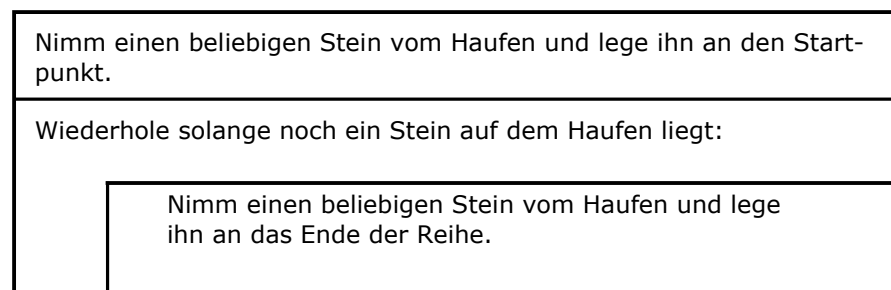
Darstellung der Sequenz:

- ❖ Nimm einen beliebigen Stein vom Haufen und lege ihn an den Startpunkt.
- ❖ Nimm einen beliebigen Stein vom Haufen und lege ihn an das Ende der Reihe.
- ❖ Nimm einen beliebigen Stein vom Haufen und lege ihn an das Ende der Reihe.
- ❖ Nimm einen beliebigen Stein vom Haufen und lege ihn an das Ende der Reihe.
- ❖ Nimm einen beliebigen Stein vom Haufen und lege ihn an das Ende der Reihe.
- ❖ Nimm einen beliebigen Stein vom Haufen und lege ihn an das Ende der Reihe.

Die fünf letzten Zeilen stellen dieselbe Anweisung dar. Sie lässt sich kürzer auch in Form einer Wiederholung darstellen:



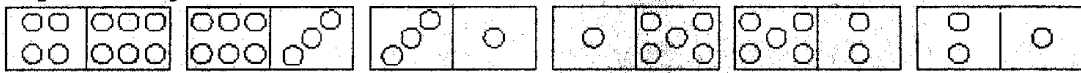
Ist die Anzahl der Dominosteine vorher nicht bekannt, lässt sich die Wiederholbedingung auch folgendermaßen formulieren:



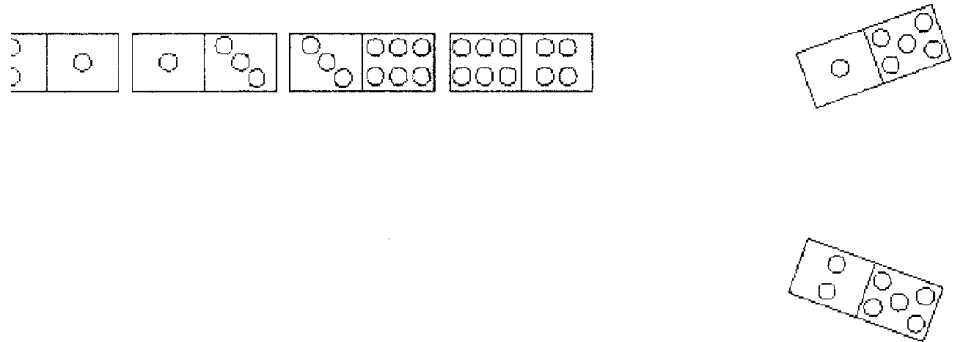
Die Dominosteine sollen nun geordnet in eine Reihe gelegt werden, und zwar so, dass Felder mit gleicher Augenzahl aneinandergelängt werden.

Mögliche Lösung.

Mögliche Lösung:

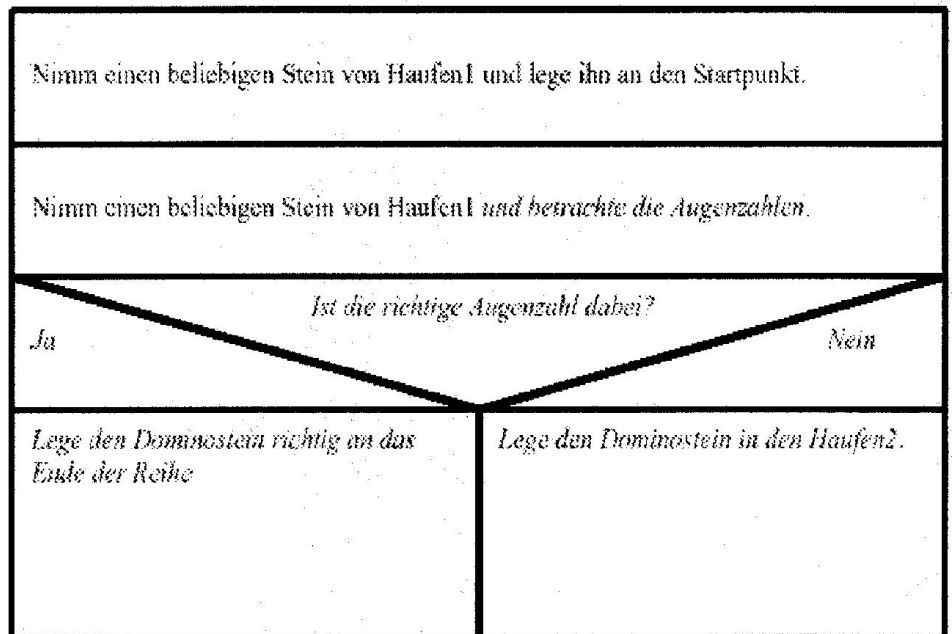


Beachte: Nicht immer gelingt es, *eine Lösung zu finden*. Der Algorithmus kann in eine *Sackgasse* führen. Beispiel mit *Sackgasse*: Keiner der beiden restlichen Dominosteine hat *die Augenzahl 4*.

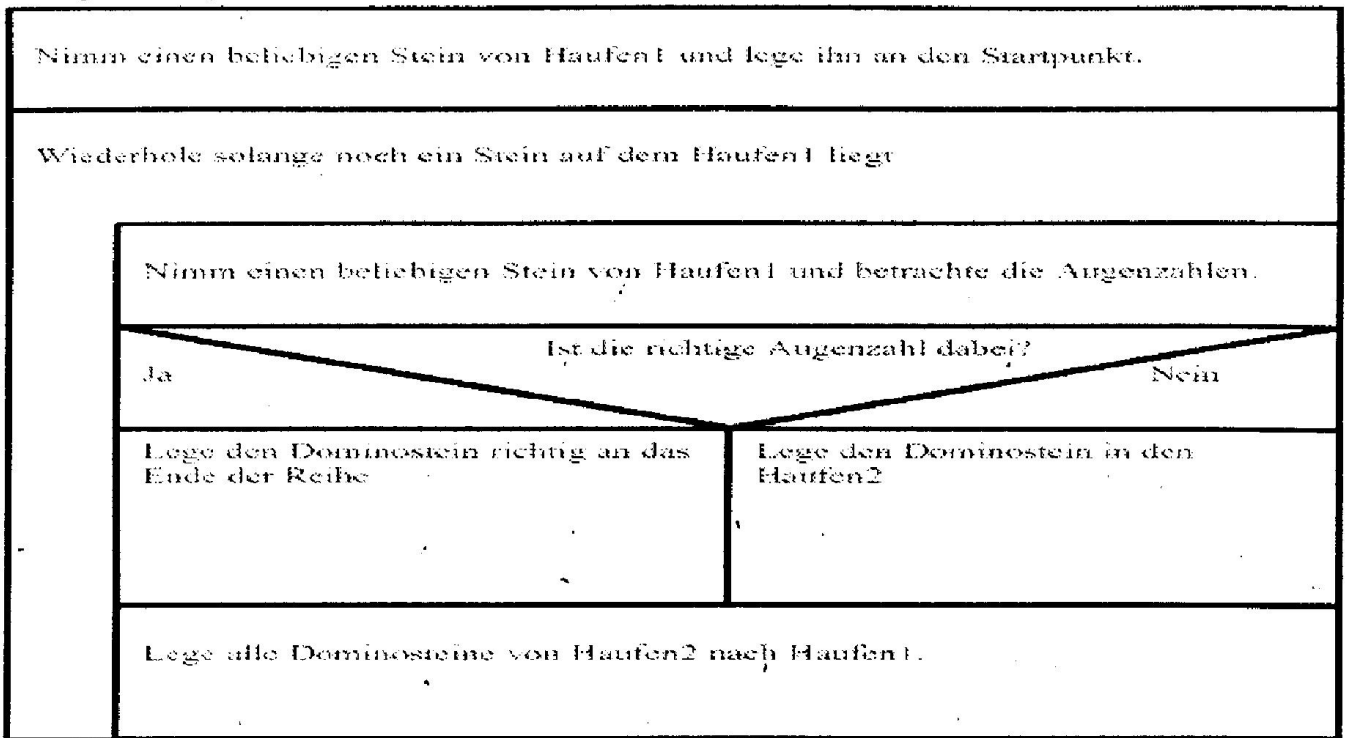


Für die "Zwischenlagerung" der entnommenen, im Moment aber unbrauchbaren, Dominosteine brauchen wir einen zweiten (zunächst noch leeren) Haufen. Die Dominosteine sollen alle auf dem ersten Haufen liegen.

Wir formulieren zunächst den Algorithmus für die ersten beiden Dominosteine.

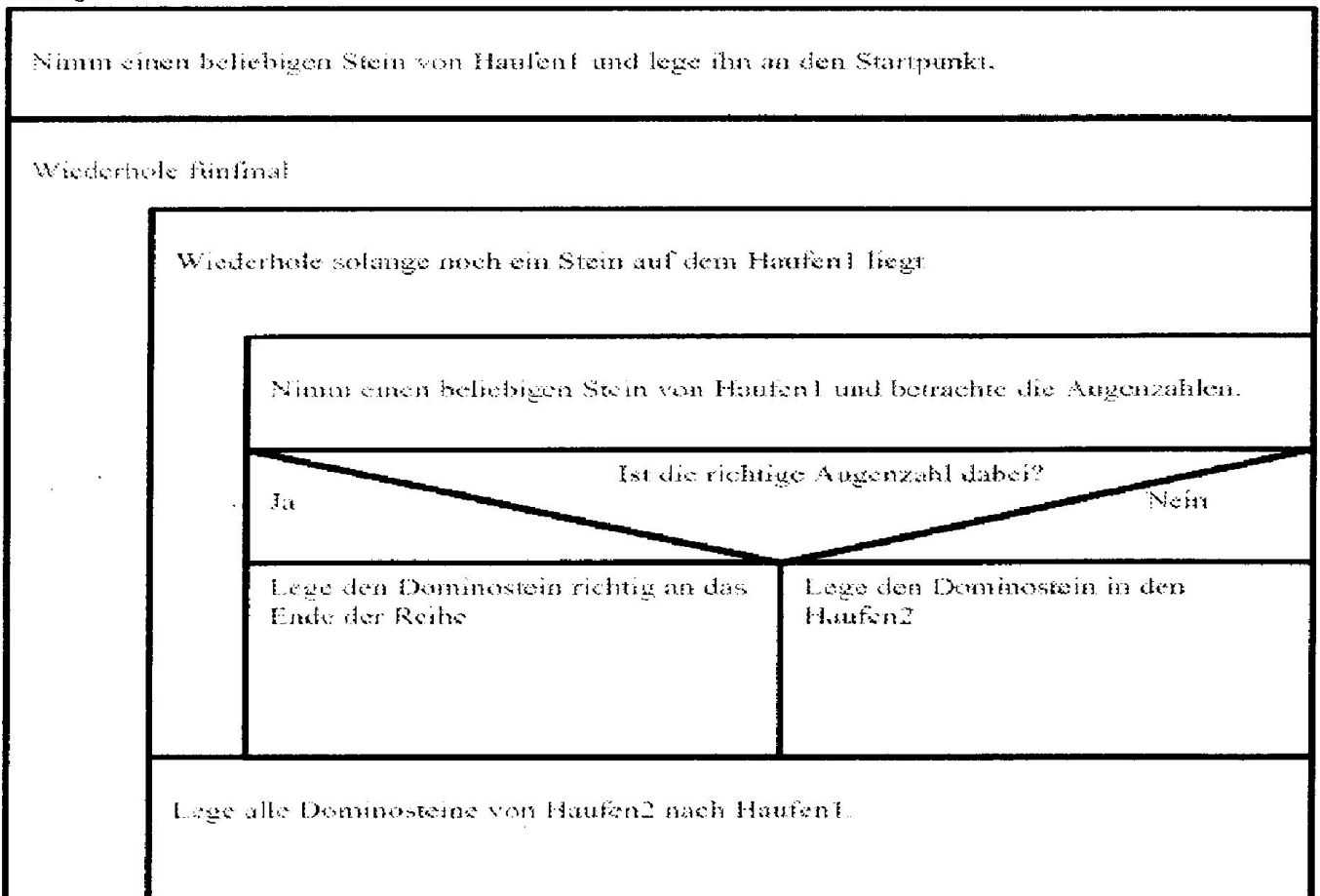


Prüfe, ob der durch das folgende Struktogramm dargestellte Algorithmus zu einer Lösung der Aufgabe führen kann. Welches Problem könnte eventuell auftreten?



Aufgabe: Zeichne das Struktogramm für den kompletten Algorithmus!

Lösung



5.2 While-Schleifen

5.2.1 Kopfgesteuerte while-Schleifen

Schleifen dienen dazu, Programmteile mehrfach mit veränderten Werten ausführen zu lassen. Schleifen werden abgebrochen, wenn eine vorgegebene Bedingung erfüllt ist, bzw. nicht mehr erfüllt ist. Bei der while-Schleife wird ein Block von Anweisungen solange ausgeführt, wie die Abbruch-Bedingung wahr ist. Das folgende Beispiel zählt zum Beispiel bis 10.

```
int i=1;
while (i<=10) {
    System.out.println(i);
    i++;
}
```

Diese Schleife kann man so übersetzen:

- ❖ Setze den Zähler i auf 1.
- ❖ Falls i < 10 ist: Gib i aus, erhöhe i um 1, springe hinter die {-Klammer
- ❖ Anderenfalls: Beende die Schleife und mache hinter } weiter

Allgemein lautet die Syntax der while-Schleife `while (bedingung) {anweisungsblock}`

Wie bei if muss die Bedingung in runden Klammern stehen.

Die Abbruchsbedingung wird jeweils überprüft, bevor der Anweisungsblock durchlaufen wird. Falls sie etwa von vornherein false ist, so wird die Schleife nie durchlaufen ('abweisende' Schleife).

Man muss sicher stellen, dass die Abbruchsbedingung auch irgendwann erfüllt wird. Sonst entsteht eine Endlosschleife, die nur durch einen Programmabbruch (Strg+C) beendet werden kann. Eine solche Endlosschleife ist etwa

```
while (true) {
    System.out.println("Bitte Strg-C drücken");
}
```

Diese Schleife druckt andauernd den gleichen Text. Das Programm muss vom Benutzer gewaltsam abgebrochen werden.

Bemerkungen

Das Struktogramm dieser 'kopfgesteuerten' Wiederholungsstruktur ist das gleiche wie für die Zählschleife.



Häufig müssen Schleifen solange durchlaufen werden, bis eine bestimmte Genauigkeit erreicht ist.

Die **for**-Schleife stellt sich nachträglich als andere Schreibweise für eine while-Schleife heraus:

```
for (i=2; i<7; i++) {System.out.println(i+1);}
bewirkt das selbe wie i=2; while (i<7) {i++; System.out.println(i);}
```

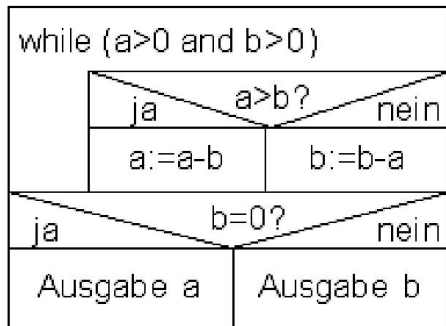
for-Schleifen werden verwendet, wenn schon vorher klar ist, wie oft die Schleife durchlaufen werden soll, bzw. von wo bis wo gezählt werden soll.

while-Schleifen werden verwendet, wenn erst zur Laufzeit des Programms entschieden wird,, nach welchem Durchlauf die Schleife abgebrochen wird.

Beispiele

Größter gemeinsamer Teiler (ggt)

Das folgende Beispiel zeigt ein Diagramm für die Berechnung des größten gemeinsamen Teilers mit dem euklidischen Algorithmus.



... und in Pascal

```
PROGRAM GGT(Input,Output);
VAR a,b: Integer;
BEGIN
  ReadLn(a,b);
  WHILE (a>0) AND (b>0) DO
    IF a>b THEN
      a:=a-b
    ELSE
      b:=b-a;
    IF b=0 THEN
      WriteLn(a)
    ELSE
      WriteLn(b)
  END.
```

Tabelle von Funktionswerten 2

Schleife zum Berechnen einer Tabelle von Funktionswerten (vom Grad- ins Bogenmaß):

```
float pi=3.1415926f; //Math.PI ist eine double-Zahl
float x;
int i=0;
while (i<=360)
{
  x = pi*i/180
  System.out.println(i+": "+x);
  i+=30; // kurz für i=i+30
}
```

Auch andere Programmiersprachen

Pascal	Java	Python
<pre>while Zahl<=100 do begin Summe := Summe+Zahl; Zahl := Zahl+1 end</pre>	<pre>while (Zahl<=100) { Summe = Summe+Zahl; Zahl = Zahl+1; }</pre>	<pre>while Zahl<=100: Summe = Summe+Zahl Zahl = Zahl+1</pre>

5.2.2 Fußgesteuerte while-Schleifen (do ... while)

Man kann die Bedingung auch ans Ende stellen: `do anweisungsblock while (bedingung);`

Hier wird die Schleife zuerst durchlaufen und dann die Bedingung getestet. Falls sie false ist, bricht die Schleife ab. Die Schleife wird also in jedem Fall mindestens einmal durchlaufen. Ansonsten unterscheidet sie sich nicht von der while-Schleife.

Beispiele

Zählen

Unser Beispiel, das von 1 bis 10 zählt, sieht dann so aus.

```
int i=1;
do
{   System.out.println(i);
    i++;
}
while (i<=10);
```

Schleifenvergleich

EINS: `for (int i=5; i<7; i++) System.out.println(i);`
`System.out.println("\n");`

`int i=5; // i war nur in der ersten Schleife definiert!`
 ZWEI: `while (i<7)`
`{`
 `i++; System.out.println(i);`
`}`

`System.out.println("\n");`

`int c=20;`
 DREI: `while (c<20)`
`{`
 `c=c+3; System.out.println(c);} // abgewiesen!`
`System.out.println("abgewiesen! c = "+c+"\n\n");`

VIER: `do c=c+3; // c war immer noch 20`
`while (c<20); System.out.println(c+"\n\n");`

Pascal	Java	Python
repeat Anweisungen until Bedingung	do Anweisungen while Bedingung	while Bedingung: Anweisungen else: Anweisungen

5.3 Unterbrechungen

Eine Schleife kann auch unterbrochen werden. Folgende Schleife wird bei $i=5$ durch **break** abgebrochen, obwohl sie eigentlich bis 9 laufen könnte:

```
for (int i=1; i<=10; i++)
{
    System.out.println(i);
    if (i==5) break;
}
```

Wenn mehrere **break**-Anweisungen in einer Schleife stehen, kann es sinnvoll sein, die Schleife nicht völlig abubrechen, sondern nur den Rest des aktuellen Schleifendurchlaufs zu überspringen. Dafür sorgt der Befehl **continue**. Danach wird die Schleifenbedingung überprüft, bei **for**-Schleifen wird vorher noch die Schleifenanweisung ausgeführt.

break verlässt einen Anweisungsblock und veranlasst das Programm am Ende des abgebrochenen Anweisungsblocks fortzufahren.

break Sprungstelle überspringt Anweisungsblöcke. Es kann aber nicht an eine beliebige Stelle des Programms (z.B. nicht zurück) gesprungen werden.

Schreibweise für die Marke Sprungstelle:

continue bricht in einer Schleife nicht die gesamte Schleife ab, sondern nur den aktuellen Schleifendurchlauf, das heißt, es wird wieder an den Anfang des Schleifenblocks gesprungen.

continue label kehrt zu der äußeren Schleife zurück, wenn sie die Markierung **Label:** hat, kann also nur in einer Schleife mit einem Label verwendet werden. Das Label ist dabei nicht das Sprungziel, sondern gibt der Schleife einen Namen.

return verlässt die aktuelle Methode und kehrt zur aufrufenden Methode zurück.

```
public class Labeltest {
    public static void main(String[] args) {
        TestSchleife:
        for (int i=1; i<100000; i++) {
            for (int j=1; j<1000000; j++) {
                System.out.println(i+" "+j);
                if (j>5) break TestSchleife;
            }
        }
    }
}
```

Sprunganweisungen

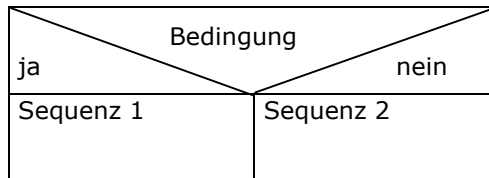
Mit **break**, **continue** und **return** können in gewissem Maße Sprünge innerhalb des Programms realisiert werden (kein "goto!"). Sie sollten so sparsam wie möglich verwendet werden, da Sprungbefehle leicht zu sehr unübersichtlichen und damit fehlerträchtigen Programmen führen können. Eine Ausnahme bildet die **switch**-Anweisung, die ohne eine **break**-Anweisung hinter den einzelnen Fallunterscheidungen nicht realisiert werden kann.

5.4 Übersicht Struktogramme

Sequenz

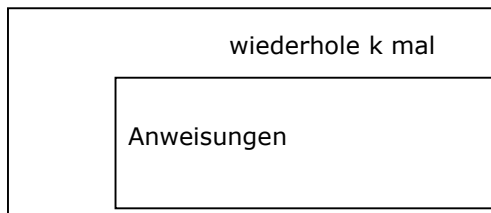


Verzweigung

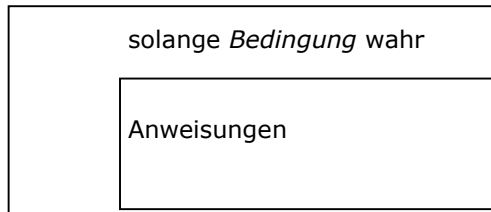


Wiederholungen

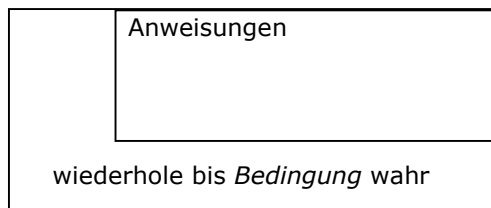
for



while

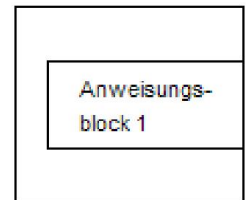


do ... while
'fußgesteuerte' Schleife



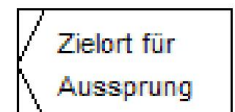
Endlos-Schleife

Sie kann allenfalls durch einen Aussprung (break) verlassen werden.



Aussprung

Der Aussprung (**break**) stellt die Beendigung eines Programmteils dar und darf nicht als das verstanden werden, was Nassi und Shneiderman mit den Struktogrammen eigentlich vermeiden wollten: Die *Sprunganweisung* (siehe oben: unbedingter Sprung Goto).



Übungsaufgaben V

-
1. Erzeuge eine Tafel der Quadratzahlen (Kubikzahlen, ...).

 2. Erzeuge eine Tabelle mit Funktionswerten für f mit $f(x) = \frac{\sin x}{\sqrt{x^2+1}}$

 3. Berechne einige Fibonacci-Zahlen. Bei dieser Zahlenfolge ist jedes Folgenglied die Summe der beiden vorhergehenden, also 1,1,2,3,5,8,13,... ($b=a+b$; $a=b-a$;)

 4. Versuche, die Zeichen über 256 im Zeichensatz auszugeben.

 5. Zahlenratespiel: Schreibe ein Programm, bei dem der Spieler eine Zahl zwischen 1 und 100 erraten muss, die sich der Computer "ausgedacht" hat. Der Spieler bekommt nach jedem Tipp zurückgemeldet, ob seine Zahl zu groß oder zu klein ist. Insgesamt soll er 5 Chancen haben.

 6. Erzeuge Lottozahlen!

 7. Schreibe ein Programm, das die Feldbezeichnungen eines Schachbretts von A1 bis H8 richtig formatiert ausgibt

 8. Was ist die Ausgabe des folgenden Programms (ohne PCI!):


```

long i=5, j=4;
while (i<=10 && j%3 !=0) {
    System.out.println(i+" "+j);
    i++; j++;
}

```

 9. Berechne die sogenannte Kollatz-Folge für den Startwert 27, bis 1 erreicht wird. Diese Folge ist definiert durch $n \rightarrow n/2$, falls n gerade ist, $n \rightarrow 3n+1$, falls n ungerade ist. Die ersten Glieder der Folge sind also 27, 82, 41, 124, 62, 31, Gib auch die Zahl der Schleifendurchläufe aus. Ändere dann den Startwert.

 10. Was passiert, wenn man auf einem Taschenrechner im Radians-Modus die Kosinustaste immer wieder drückt? Schreibe eine Endlosschleife, die sich nur mit STRG+C abbrechen lässt, und eine Schleife die sich selbst unterbricht, wenn die Iteration bei einer Zahl stehen bleibt.

 11. Wiederhole Aufgabe 10 mit der Sinustaste und dem Startwert 1.

 12. Berechne e mit Hilfe der Exponentialsumme $1+1/1!+1/2!+1/3!+\dots$ auf 14 Dezimalen. Vergleiche mit Math.E. **Abbruch nach 16 Schritten**

 13. Berechne $(1+1/n)^n$ für wachsende Werte von n . Wie groß muss man n wählen, damit dieser Ausdruck von e (=Math.E) weniger als 0.01 entfernt ist? ($n=135$)

 14. Nimm eine beliebige Zahl zwischen 0 und 1 und multipliziere diese Zahl mit 2. Wenn Sie größer als 1 wird, subtrahiere 1. Führe diesen Prozess in einer Schleife fort, bis die Zahl 1 erreicht ist. Zähle dabei die Schleifendurchläufe. Warum endet diese Schleife immer?

15. Was ergibt $(1/11)*11-1$, wenn man es mit `double`-Werten auswertet? Bestimme an Stelle von 11 die erste Zahl, bei der dieser Ausdruck nicht 0 ergibt.

16. a) Berechne für $a=222$ und $b=36$ das Ergebnis bei wiederholter Subtraktion $a=a-b$. Zähle auch die Schleifendurchläufe.

b) Welches Problem löst dieser Algorithmus?

c) Subtrahiere weiter jeweils die kleinere Zahl von der größeren. Welche Bedeutung hat das letzte Ergebnis, das größer als Null ist?

d) In welchem Fall wird der 'Solange'-Block übersprungen?

17. Kaprekar-Zahlen

1. Starte mit einer beliebigen vierstelligen Zahl >1000
2. Stelle die Ziffern so um, dass ein Mal die größte und ein mal die kleinste Zahl entsteht.
3. Subtrahiere die beiden Zahlen voneinander.

Wiederhole 2. und 3. bis sich die Zahl nicht mehr ändert.

18 Welchen Wert hat c bei $c=16$; `while (c<20) c=c+3`; nach Schleifenende 22!?

Welchen Wert hat dann c bei $c=16$; `do c=c+3; while (c<20)`; 22!?

Beginne in beiden vorigen Beispielen mit $c=20$ statt $c=16$!

19. Schreibe mit allen drei Schleifentypen ein Programm, das die ersten 100 Quadratzahlen zusammenzählt.

6. Arrays

Für Vektoren und Tabellen möchte man manchmal eine beliebige Anzahl von Variablen gleichen Typs gleichzeitig erzeugen und verwenden. Solche *Felder*, in Programmiersprachen *Arrays* genannt, werden mit eckigen Klammern `[]` gekennzeichnet. Das Array an sich muss mit **new** bereitgestellt werden. Im folgenden Beispiel wollen wir 1000 Variablen vom Typ **double** erzeugen, die in einem Array zusammengefasst werden.

Man kann ein Array auch schon bei der Deklaration mit Werten versehen. In diesem Fall ist kein **new** notwendig.

Beispiel: `int n[]={1,2,3,4,5};`

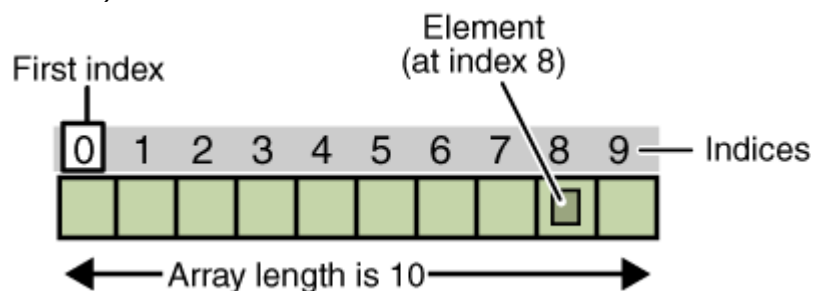
Die Größe des Arrays ergibt sich aus der Anzahl der angegebenen Werte.

am kürzesten: **double**
`a[]=new double[1000];`

Die Größe eines Arrays wird bei der Erzeugung festgelegt, und kann nicht mehr geändert werden. Mit `a.length` kann sie aber im Programm verwendet werden. Mit obigem Befehl stehen also 1000 **double**-Variablen zur Verfügung. Der Zugriff auf eine einzelne dieser Variablen erfolgt durch Indizierung. Dabei wird der gewünschte Index in eckigen Klammern angegeben. Als Beispiel speichern wir die Quadratzahlen 1^2 , 2^2 , ..., 1000^2 in dem Array ab.

```
for (i=0; i<1000; i++) a[i]=i*i;
```

Vorsicht: Beachte, dass die Nummerierung im Array immer mit 0 beginnt! Das letzte Element hat also den Index 999 (Größe minus 1).



Die allgemeine Syntax einer indizierten Array-Variablen ist `arrayname[index]`, wobei **index** ein Ausdruck vom Typ **int** sein muss. Eine solche indizierte Variable kann wie jede andere Variable des entsprechenden Typs (im Beispiel **double**) verwendet werden.

Ein anderes Beispiel, bei dem nun eine Variable des Arrays sowohl rechts als auch links vom Zuordnungszeichen erscheint.

```
a[0]=1.0;
```

```
for (i=1; i<1000; i++) a[i]=1.01*a[i-1];
```

Dieses Beispiel setzt jedes Element des Arrays auf das 1.01-fache des vorherigen, beginnend mit 1. Man muss also das erste Element setzen und dann die Restlichen in einer Schleife bearbeiten.

Fehlermeldungen

Falls ein Index außerhalb der Grenzen 0 bis $n-1$ liegt, wird eine **index-out-of-bounds exception** erzeugt. Bei unseren einfachen Test-Programmen führt dies zum Programmabbruch mit einer Fehlermeldung. Wird die Initialisierung des Arrays mit **new** weggelassen, so hat die Variable den vordefinierten Inhalt null. Jeder Versuch, auf das Array zuzugreifen, erzeugt dann eine sogenannte **NullPointerException**.

Statistik von Zufallszahlen

Wir erzeugen 10000 Zufallszahlen, bestimmen den Mittelwert und zählen, wie viele Zahlen in jedem Intervall 0 bis 0.1, 0.1 bis 0.2 etc. liegen.

Es wird zunächst Platz für 10000 Zahlen in einem Array geschaffen. Dann werden 10000 Zufallszahlen erzeugt und in dem Array abgespeichert. Beim Hantieren mit dem Array muss man beachten, dass der Index von 0 bis 9999 laufen muss.

Ein weiteres Array '**vert**' (Verteilung) der Größe 10 nimmt schließlich die Zähler auf. **vert[0]** soll die Anzahl der Zufallszahlen zwischen 0 und 0.1 enthalten, usw. Der Index des Zählerintervalls wird aus der Zufallszahl durch Multiplikation mit 10 und Abschneiden des Nachkomma-Anteils bestimmt.

```
int n=10000;           // Anzahl der Zufallszahlen
double Zahl[]=new double[n]; // erzeugt Platz fuer n Zahlen
int i;                // Schleifenzähler

// Zufallszahlen:
for (i=0; i<n; i++) Zahl[i]=Math.random();

// Mittelwert:
double sum=0, mittel;
for (i=0; i<n; i++) sum+=Zahl[i]; // berechne Summe
mittel=sum/n;
System.out.println("Mittelwert: "+mittel);

// Verteilung:
int m=10, j;
int vert[]=new int[m];           // Platz fuer m=10 Zaehler in 10 Intervallen
for (j=0; j<m; j++) vert[j]=0;   // alle mit 0 initialisieren
for (i=0; i<n; i++)              // zaehlen
{
    j=(int)Math.floor(Zahl[i]*m); // in welchem Intervall liegt Zahl[i]?
    // if (j>=m) j=m-1;           // falls Zahl[i]=1.0
    vert[j]++;                   // erhoehe Zaehlernummer
}

// Ausgabe:
for (j=0; j<m; j++)              // Ausgabe der Zaehlungen
    System.out.println((j+1)+" : "+vert[j]);
```

Einige Dinge sind erklärungsbedürftig.

Wir verwenden **Math.random()** zur Erzeugung von Zufallszahlen zwischen 0 und 1. Die Zufallszahlen werden in einem Array abgespeichert, so dass wir mehrere Operationen mit den Zahlen vornehmen können.

Der **Mittelwert** ist die Summe der Zahlen, dividiert durch die Anzahl der Zahlen.

Um die Anzahl der Zahlen in den 10 Teilintervallen zu zählen, muss man für jedes Teilintervall einen Zähler einrichten. Nötig ist also ein Array aus 10 **int**-Variablen.

Das Intervall, in dem die Zahl x liegt wird folgendermaßen berechnet: x wird mit 10 multipliziert und liegt dann im Intervall 0 bis 10. Dann wird der ganzzahlige Anteil genommen. Dies erledigt die Funktion **Math.floor()**, die einen **double**-Wert als Ergebnis hat. Der Wert muss daher noch nach **int** umgewandelt werden. Es entsteht eine der Zahlen 0,...,9.

Man beachte, dass die Größe des Arrays im **new**-Befehl auch durch eine Variable oder einen Ausdruck angegeben werden kann. Das Programm erzeugt z.B. die folgende Ausgabe

```
Mittelwert : 0.5013022861887491
1: 1008; 2: 1004; 3: 994; ...
```

Diese Ausgabe ist natürlich nicht jedes Mal gleich, da der Zufallsgenerator bei jedem Lauf initialisiert wird.

Bubble-Sort

Das folgende Programm erzeugt Zufallszahlen und sortiert sie. Der Sortieralgorithmus ist der denkbar schlechteste, aber auch der einfachste. Daher ist dieses Programm nur für kleine Datenmengen brauchbar.

Der Algorithmus funktioniert so: Die Zahlen **R[1]** bis **R[n-1]** werden solange nach vorne getauscht, bis sie an der richtigen Stelle stehen. Dadurch steigen die kleineren Zahlen nach oben, zum Listenbeginn auf, wie Blasen im Bier.

1. Lasse j vom zweiten Index (1) bis zum letzten Index (n-1) laufen.
2. Schau nach, ob die j-te Zahl kleiner als die (j-1)-te ist. Wenn ja vertausche die beiden Zahlen. Wenn nein, so stehe die Zahl richtig und wir gehen in 1. zum nächsten Index.
3. Falls vertauscht wurde, so wiederhole den Schritt 2. mit der (j-2)-ten und der (j-1)-ten Zahl, und dann immer weiter bis zur 2. und 1. Zahl.

Durch 2. und 3. wird die j-te Zahl an die richtige Stelle sortiert.

```
{  int n=20;
    double R[]=new double[n];
    double h;
    int i,j;

    // Erzeuge Zufallszahlen:
    for (i=0; i<n; i++) R[i]=Math.random();

    // Sortiere:
    for (i=1; i<n; i++)
    {  j=i;                // Sortiere R[i] ein
        while (j>0)      // solange es nicht ganz vorne steht
        {  if (R[j-1]>R[j]) // steht noch falsch
            {  h=R[j]; R[j]=R[j-1]; R[j-1]=h; // vertauschen
            }
            else break;   // steht richtig
            j--;
        }
    }

    // Ausgabe:
    for (i=0; i<n; i++)
        System.out.println(R[i]); // 20 geordnete Zufallszahlen
}
```

Übungsaufgaben VI

1. Schreibe ein Programm zur Notenberechnung. Es sollen nacheinander die Noten von 4 Klausuren eingegeben werden, anschließend werden alle Noten und der Durchschnitt ausgegeben.

2. Schreibe ein Programm zur Addition von zwei Vektoren. Jeder Vektor wird dabei als Array dargestellt. Erweitere dann das Programm um die Berechnung des Skalarprodukts und der Beträge der Vektoren.

3. Erzeuge 1000 Zufallsvariablen, die in einem Array zwischengespeichert werden, und bestimme die größte davon. Wie lässt sich die Aufgabe ohne Array lösen?

4. Wie 3. Jedoch ist diesmal nach der zweitgrößten Zahl gefragt.

5. Versuche 1000 (10000) Zahlen mit dem Sortieralgorithmus zu sortieren.

6. Erzeuge das Pascalsche Dreieck:


```

                1
              1 1
            1 2 1
          1 3 3 1
        1 4 6 4 1
            
```

 Speichere die Zeilen in einem einfachen Vektor ab und gib diesen Vektor in einer Zeile aus (für die ersten 20 Zeilen).

7. Speichere wieder eine große Anzahl von Zufallszahlen und teste, wie oft die nächste größer als die vorhergehende ist. Teste, wie oft die nächste Zahl größer als die beiden vorhergehenden ist. Entspricht das Ergebnis Deinen Erwartungen?


8. Schreibe ein Programm, das eine Lottoziehung (6 aus 49 mit Zusatzzahl) simuliert. Wichtig ist natürlich, keine Zahlen doppelt zu ziehen. Eine einfache Strategie dazu ist, solange eine Zufallszahl zu erzeugen, bis eine neue Zahl entsteht. Sortiere dann die Zahlen.

9. Schreibe die Zahlen 1 bis n in ein Array und mische dieses Array. Dazu vertausche $k=2$ bis $k=n$ mit einer Zahl an einer zufälligen Stelle von 1 bis k.

10. Führe die vorige Aufgabe für ein großes Array durch und teste, wie viele Zahlen an derselben Stelle stehen bleiben.

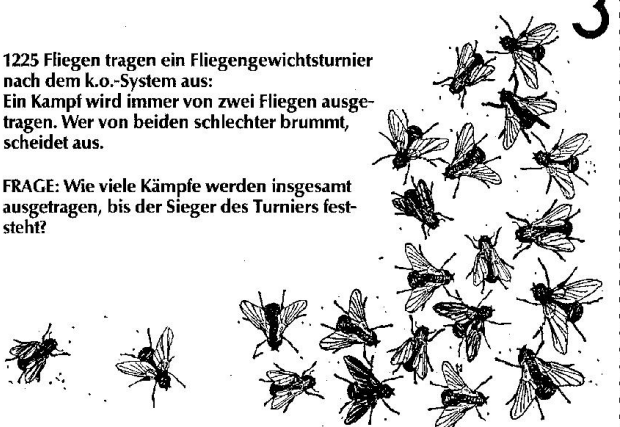
11

Bankier Dagobert schätzt sein Vermögen an Pfennigstücken auf etwa 1 bis 2 Milliarden Stück.
 Aus Langeweile legt er alle Pfennige einmal als Quadrat und einmal in Dreiecksform. Beide Male verwendet er alle Pfennige, ohne daß welche übrigbleiben.
 FRAGE: Wieviel Pfennige hat er? Antwort – Quersumme dieser Zahl.



2

1225 Fliegen tragen ein Fliegengewichtsturnier nach dem k.o.-System aus: Ein Kampf wird immer von zwei Fliegen ausgetragen. Wer von beiden schlechter brummt, scheidet aus.
 FRAGE: Wie viele Kämpfe werden insgesamt ausgetragen, bis der Sieger des Turniers feststeht?



3

7. Strings

Ein String ist ein Objekt, bestehend aus (im Allgemeinen) mehreren Zeichen. Im Gegensatz dazu bestehen die anderen Datentypen nur aus einzelnen Zeichen oder Zahlen. In Variablen von diesen Datentypen kann man 'Zeichenketten' speichern und unter den Namen der Variablen auch wieder ansprechen bzw. aufrufen. Strings sind für viele nicht-mathematischen Programmteile entscheidend wichtig.

Strings sind keine elementaren Datentypen, sondern verhalten sich wie Klassen. Ein Text soll ja nicht für jeden Buchstaben eine Variable verbrauchen, sondern als Ganzes gespeichert werden. Damit hängt auch zusammen, dass Strings durch einen Konstruktor hergestellt werden.

Es gibt mehrere Möglichkeiten der **Deklaration** und **Initialisierung** einer Stringvariablen:

```
String nachname; nachname = "Herz";
```

Oder zusammengefasst: **String nachname = "Herz";**

Allgemeiner, weil ähnlich auch für andere Aufgaben einsetzbar:

```
String nachname = new String();  
nachname = "Java";
```

Dieser Konstruktor verdeutlicht auch, was passieren muss:

1. Es soll ein neues Objekt der Klasse **String** erzeugt werden.
2. Dieses Objekt soll unter dem Bezeichner **nachname** angesprochen werden.
3. Und diesem String wird der Wert **'Herz'** zugeordnet.

zusammengefasst: **String nachname = new String("Herz");**

String-'Werte' werden in Anführungszeichen eingeschlossen.

Beispiel: **String satz="Dies ist ein Teststring.";**

satz = Console.readString() erwartet die Eingabe eines Strings (mit **Console.class**).

Die wichtigste Operation mit Strings ist die Addition **+**. Diese hängt zwei Strings hintereinander. Man kann auch elementare Datentypen zu Strings 'addieren'. Diese Werte werden dann intern erst in Strings umgewandelt - nur dadurch kann man numerische Zahlen mit Hilfe von Ziffern ausgeben!

```
double x = 3.14;  
System.out.println("Pi ist ungefähr "+ x);
```

oder **System.out.println(vorname + " " +nachname);**

Will man einen String mit einem anderen verbinden und das Ergebnis in einem String abspeichern, verwendet man die Methode **concat**, die jeder String hat. Als Argument erhält sie den zweiten String, der mit dem ersten verbunden werden soll.

```
class Stringtest {  
    public static void main(String[] args) { String vorname = "Eva";  
        String nachname = new String("Herz"); String zeile = "Name: ";  
        zeile =zeile.concat(vorname);  
        zeile =zeile.concat(" ");  
        zeile =zeile.concat(nachname);  
        System.out.print(zeile);  
        System.out.println(zeile.length());  
    }  
}
```

Die letzte Methode, in diesem Beispiel für das Objekt **zeile**, ist die Methode **length**. Sie ermittelt die Länge der Zeichenkette.

Das Ergebnis unseres Programms lautet daher also: **Name: Eva Herz 18**

Übungsaufgaben VII

1. Schreibe ein Programm, welches die Variablen vorname und nachname einliest und z.B. ausgibt: "Guten Tag Angela Merkel". Verwende `Console.readString()`!