

# Inhaltsverzeichnis

Kapitel 1: Allgemeines zum Informatikkurs.....	3
Kapitel 1.1: Anforderungsbereiche.....	3
Kapitel 1.2: Klausuren.....	5
Kapitel 1.3: Sonstige Mitarbeit.....	5
Kapitel 2: Was ist Informatik, womit beschäftigt sich die Informatik.....	7
Kapitel 2.1: Einige Begriffserläuterungen:.....	7
Kapitel 2.2: Teilgebiete der Informatik.....	7
Kapitel 2.2.1: Die Theoretische, Technische und Praktische Informatik.....	7
Kapitel 2.3: Themengebiete in der Einführungsphase.....	8
Kapitel 2.4: Computer und Programme; Algorithmus.....	9
Kapitel 2.5: Programme lesen und ausführen.....	9
Kapitel 2.6: Programmiersprachen.....	9
Kapitel 2.7: Der Compiler.....	10
Kapitel 2.8: Der Interpreter.....	10
Kapitel 2.9: Compiler vs. Interpreter.....	10
Kapitel 2.10: Allgemeines über Java.....	11
Kapitel 2.11: Die Vorteile von Java-Bytecode.....	11
Kapitel 3: Erste Schritte mit Java.....	12
Kapitel 3.1: Vorbereitung zum Programmieren.....	12
Kapitel 3.2: Das Java Development Kit (JDK).....	12
Kapitel 3.3: Die Programmierumgebung → Installation des Java-Editors (Zu Hause).....	12
Kapitel 3.3.1: Systemvariablen anpassen (Zu Hause).....	13
Kapitel 3.4: Die Alternative Möglichkeit die Systemvariablen über den „JavaEditor“ anpassen (Zu Hause).....	14
Kapitel 3.5: Die Ordnerstruktur anlegen (In der Schule).....	15
Kapitel 3.6: Den ersten Java-Quellcode erstellen , speichern und ausführen.....	15
Kapitel 3.7: Strukturelemente und Anweisungen.....	17
Kapitel 3.7.1: Die Rolle der Klasse.....	17
Kapitel 3.7.1.1: Die HalloJava-Klasse.....	17
Kapitel 3.7.1.2: Aufbau einer Methode.....	17
Kapitel 3.7.1.3: Die Rolle der main-Methode.....	18
Kapitel 3.7.1.4: Bedeutung der Schlüsselwörter:.....	18
Kapitel 3.7.1.5: Die Ausgabeanweisung.....	19
Kapitel 3.7.1.6: Kommentieren und Einrücken.....	20
Kapitel 3.7.2: Variablen und Konstanten.....	21
Kapitel 3.7.2.1: Die Rolle der Variablen.....	21
Kapitel 3.7.2.2: Variablen anlegen.....	22
Kapitel 3.7.2.3: Einfache Datentypen.....	22
Kapitel 3.7.2.4: Der Datentyp char.....	23
Kapitel 3.7.2.5: Der Datentyp boolean.....	23
Kapitel 3.7.2.6: Der Datentyp String.....	23
Kapitel 3.7.2.7: Umwandlungen von Datentypen.....	24
Kapitel 3.7.2.8: Explizite Typumwandlung.....	24
Kapitel 3.7.2.9: Übersicht zu impliziten Typumwandlungen.....	24
Kapitel 3.7.2.10: Arithmetische Operatoren.....	25
Kapitel 3.7.3: Verkettung von zwei Strings (Z.B. zwei Wörtern).....	26
Kapitel 3.7.3.1: Inhalte von Variablen ändern.....	29
Kapitel 3.7.3.2: Konstanten.....	29
Kapitel 3.7.3.3: Variablen und der Ausgabebefehl.....	30
Kapitel 3.8: Einfache Benutzereingaben vom Programm einlesen lassen .....	30
Kapitel 3.9: Kontrollstrukturen.....	31
Kapitel 3.10: Verzweigungen.....	31
Kapitel 3.10.1: Vergleichsoperationen.....	31

Kapitel 3.10.2: Logische Operatoren.....	32
Kapitel 3.10.3: <i>Klassische Verzweigung mit if</i> .....	33
Kapitel 3.11: Verzweigung mit switch.....	38
Kapitel 4: Beispielklausur Nr. 1: Thema: Primitive Datentypen, Variablen, Kontrollstrukturen I.....	42
Kapitel 5: Schleifen.....	45
Kapitel 5.1: Die for-Schleife.....	47
Kapitel 5.2: Die while-Schleife.....	47
Kapitel 5.3: Schleifen mit do while.....	48
Kapitel 5.4: Schleifensteuerung durch Laufbedingungen.....	49
Kapitel 5.5: Terminierung einer Schleife.....	49
Kapitel 5.6: Laufbedingung (Terminierung) bei Verwendung von Gleitkommazahlen.....	49
Kapitel 5.7: Schleifen ohne Schleifenkörper.....	49
Kapitel 5.7.1: Aufgabe: Projekt Automat.....	57
Kapitel 6: Felder (Arrays) in Java.....	58
Kapitel 6.1: Deklaration und Zuweisung.....	59
Kapitel 6.2: Zugriff auf Elemente eines Feldes.....	60
Kapitel 7: Zweidimensionale Felder (Arrays) in Java.....	63
Kapitel 7.1: Deklaration eines zweidimensionalen Arrays.....	63
Kapitel 7.2: Anlegen eines zweidimensionalen Arrays.....	63
Kapitel 7.3: Speichern und Auslesen bei zweidimensionalen Arrays.....	63
Kapitel 8: <i>Methoden bzw. Funktionen in Java</i> .....	66
Kapitel 8.1: Was sind Methoden?.....	66
Kapitel 8.2: Vorteile von Methoden.....	66
Kapitel 8.3: Eine Methode erstellen.....	66
Kapitel 8.3.1: Die Methode (Funktion) gibAus.....	67
Kapitel 8.4: Deklaration von Methoden (Funktionen).....	67
Kapitel 8.5: Aufruf von Methoden (Funktionen).....	71
Kapitel 8.6: Arrays und Methoden.....	74
Kapitel 8.7: Der Gültigkeitsbereich von Variablen.....	80
Kapitel 9: Klassen und Objekte, Instanzieren von Objekten einer Klasse.....	88
Kapitel 9.1: Zugriffsmodifikatoren (Sichtbarkeit).....	93
Kapitel 9.2: Klassenkarten.....	94
Kapitel 10: Anhang.....	95

## Kapitel 1: Allgemeines zum Informatikkurs

Als Informatiklehrkräfte können wir gar nichts anderes sagen, als dass die Informatik extrem wichtig zum Verständnis des Universums und der Welt in der wir leben, ist. Das bringt jedoch eines mit sich: Informatik ist nicht leicht und sehr arbeitsintensiv...

Was heißt das nun für Sie?

1. Pause machen? Fehlanzeige! Sie müssen immer am Ball bleiben.
2. Aussitzen bringt nichts! Lassen Sie sich drauf ein!
3. Alles unmöglich? Alles viel zu schwierig? Nur, wenn Sie die Punkte 1 und 2 nicht beachten.
4. Informatik macht keinen Spaß? Doch, wenn Interesse da ist und Sie sich auf die Punkte 1 und 2 einlassen!
5. Alles unmöglich zu schaffen? Nein, das haben schon Tausende vor Ihnen geschafft. Und die waren nicht klüger oder intelligenter als Sie.

Kurz gesagt: Setzen Sie sich ein realistisches Ziel und arbeiten Sie selbständig darauf hin! Nichts ist unmöglich!

Das Ziel, was Sie sich setzen, kann zum Beispiel die Note sein, die Sie am Ende des Schuljahres erreichen möchten. Die Fachschaft Informatik hat dabei einen Kriterienkatalog entwickelt, aus dem hervorgeht, was Sie für welche Note tun müssen. Dieser Katalog sehen Sie auf den nächsten Seiten.

### Kapitel 1.1: Anforderungsbereiche

Die Leistungsbewertung in der Sekundarstufe II bezieht sich auf die im Kernlehrplan benannten Kompetenzbereiche und unterscheidet dabei die drei verschiedenen Anforderungsbereiche.	
Anforderungsbereich I (Reproduktion)	<ul style="list-style-type: none"><li>• Sie geben bekannte Argumentationen wieder, bestätigen oder widerlegen eine Aussage durch einschrittiges logisches Schließen. Sie geben Begründungen in bekannten Zusammenhängen wieder.</li></ul>
Anforderungsbereich II (Reorganisation und Transfer)	<ul style="list-style-type: none"><li>• Sie begründen mithilfe eigener Argumente oder Argumentationsketten.</li><li>• Sie bewerten fachliche Darstellungen und die Eignung von Darstellungsformen, beurteilen informatische Sachverhalte anhand gegebener Kriterien.</li><li>• Sie bewerten Informatiksysteme unter z.B. fachlichen, ethischen, ökologischen, ökonomischen und rechtlichen Aspekten.</li></ul>
Anforderungsbereich III (Reflexion und Problemlösung)	<ul style="list-style-type: none"><li>• Sie begründen komplexere informatische Sachverhalte und entwickeln dafür Argumentationsketten.</li><li>• Sie beurteilen bzw. bewerten informatische Sachverhalte – auch in authentischen Darstellungen – oder die Eignung von Informatiksystemen anhand selbst gewählter fachlicher sowie für die Nutzung relevanter Kriterien.</li><li>• Sie bewerten den eigenen oder gemeinsamen Arbeitsprozess und dessen Ergebnisse und ziehen Schlüsse für ihr zukünftiges Handeln.</li></ul>

Die folgende Übersicht zeigt Beispiele, wie Leistungen den Anforderungsbereichen zugeordnet werden können:

#### Inhaltsfeld „**Information und Daten**“

- Sie unterscheiden zwischen Zeichen, Daten und Information sowie zwischen Syntax und Semantik (I).
- Sie analysieren Daten hinsichtlich ihrer Struktur (I-II),
- Sie bilden Information als Daten mit Datentypen und in Datenstrukturen ab (I).
- Sie verwenden, modellieren und implementieren Operationen auf statischen und dynamischen Datenstrukturen (I-II).
- Sie erstellen zu einem Realitätsausschnitt ein Datenmodell und implementieren es als Datenbank (II).
- Sie untersuchen und organisieren Daten unter Beachtung von Redundanz, Konsistenz und Persistenz (II-III).
- Sie verwenden eine Abfragesprache zur Anzeige und Manipulation von Daten und interpretieren die Daten (II).
- Sie verwenden, modellieren und implementieren Operationen auf komplexen Datenstrukturen (III).
- Sie entwickeln zu einem Ausschnitt der Lebenswelt mit komplexen Beziehungen eine Datenbank (III).

#### Inhaltsfeld „**Algorithmen**“

- Sie verwenden algorithmische Grundbausteine (Sequenz, Alternative, Wiederholung) und implementieren diese mithilfe einer Programmiersprache (I).
- Sie analysieren gegebene Programme hinsichtlich der Grundkonzepte, einschließlich Variable, Referenz, Schachtelung und funktionaler Zerlegung (II).
- Sie entwerfen Algorithmen und stellen sie in geeigneter Form dar (I-II).
- Sie wenden Modularisierung zur Strukturierung von Algorithmen und bei deren Implementierung an (II-III).
- Sie verwenden Softwarebibliotheken oder bereitgestellte Module bei der Implementierung von Algorithmen (I-II).
- Sie testen und überarbeiten Programme systematisch (II).
- Sie modellieren und implementieren iterative und rekursive Algorithmen und Datenstrukturen (II-III).
- Sie vergleichen und beurteilen Algorithmen zur Lösung eines Problems, unter anderem hinsichtlich der Effizienz (III).
- Sie analysieren an Beispielen die Komplexität von Algorithmen und beurteilen die praktischen und theoretischen Grenzen der Algorithmisierung (III).

#### Inhaltsfeld „**Sprachen und Automaten**“

- Sie vergleichen formale mit natürlichen Sprachen, untersuchen den Zusammenhang zwischen einer Grammatik und ihrer Sprache (I-II).
- Sie leiten Wörter einer Sprache ab und stellen Ableitungsbäume dar (II).
- Sie verwenden Sprachdefinitionen (z.B. Grammatiken, Syntaxdiagramme) zur Analyse, Beschreibung und Entwicklung formaler Sprachen (II).
- Sie überführen Grammatiken in endliche Automaten und umgekehrt (II-III).
- Sie erläutern den Zusammenhang zwischen Grammatiken, Sprachen und Automaten (III).
- Sie analysieren und implementieren Programme zu Problemstellungen auf Kellerautomaten, Turingmaschinen oder Registermaschinen (III).
- Sie erläutern prinzipielle und praktische Grenzen der Berechenbarkeit (III).

#### Inhaltsfeld „**Informatiksysteme**“

- Sie erläutern prinzipielle Funktionsweisen und das Zusammenwirken wesentlicher Hardware-, Software- und Netzwerkkomponenten (I).
- Sie beschreiben und erklären wesentliche Schichten und Komponenten der Architektur gegebener Informatiksysteme und die damit verbundenen Prozesse (I-II).
- Sie entwickeln ein Netzwerk mithilfe geeigneter Strukturierungs- und Darstellungsmethoden, verwenden den objektorientierten Ansatz, indem sie Klassen mit ihren Attributen, Methoden und Beziehungen modellieren und implementieren (II-III).
- Sie analysieren die Kommunikation und die Datenhaltung in vernetzten Systemen und beurteilen diese auch unter den Gesichtspunkten des Datenschutzes und der Datensicherheit (II-III).

- Sie wenden Konzepte und Methoden der Softwareentwicklung zur Gestaltung und Entwicklung von Informatiksystemen an, auch unter Berücksichtigung von Aspekten der Softwareergonomie (III).
- gestalten Informatiksysteme auf Basis von Qualitätskriterien (z.B. Robustheit, Wiederverwendbarkeit, Korrektheit, Effizienz, Komplexität) (III).

#### Inhaltsfeld „Informatik, Mensch und Gesellschaft“

- Sie analysieren und beschreiben Wechselwirkungen zwischen Informatiksystemen, Individuen und Gesellschaft (I).
- Sie beschreiben Chancen, Risiken und Missbrauchsmöglichkeiten von Informatiksystemen, diskutieren und bewerten wesentliche Aspekte des Datenschutz- und Urheberrechts anhand von Anwendungsfällen (II-III).
- Sie beurteilen und bewerten die gesellschaftlichen Folgen der Einführung und Nutzung von Informatiksystemen, verwenden und beschreiben Verfahren zur Sicherung von Vertraulichkeit, Authentizität und Integrität von Daten, ziehen Rückschlüsse für das eigene Verhalten beim Einsatz von Informatiksystemen (II-III).
- Sie analysieren und beurteilen Verfahren zur Sicherung von Vertraulichkeit, Authentizität oder Integrität von Daten in konkreten aktuellen Anwendungskontexten (III).
- Sie konzipieren Maßnahmen zur Realisierung von Datensicherheit für konkrete Anwendungsfälle, insbesondere Zugriffskontrolle (III).

### Kapitel 1.2: Klausuren

Im ersten Halbjahr der EF wird eine, im zweiten Halbjahr werden zwei Klausuren geschrieben.

Die Notenfestsetzung erfolgt in der Regel nach folgendem Schlüssel. In Ausnahmefällen kann begründet davon abgewichen werden.

<i>Leistungsbeurteilung</i>	<i>Erreichte Hilfspunktzahl in %</i>
<i>sehr gut</i>	<i>≥85 bis 100</i>
<i>gut</i>	<i>≥70 bis 85</i>
<i>befriedigend</i>	<i>≥55 bis 70</i>
<i>ausreichend</i>	<i>≥40 bis 55</i>
<i>mangelhaft</i>	<i>≥20 bis 40</i>
<i>ungenügend</i>	<i>&lt;20</i>

### Kapitel 1.3: Sonstige Mitarbeit

Folgende Aspekte können bei der Leistungsbewertung der sonstigen Mitarbeit eine Rolle spielen (die Liste ist nicht abgeschlossen):

- Sicherheit, Eigenständigkeit und Kreativität beim Anwenden fachspezifischer Methoden und Arbeitsweisen
- Verständlichkeit und Präzision beim zusammenfassenden Darstellen und Erläutern von Lösungen einer Einzel-, Partner-, Gruppenarbeit oder einer anderen Sozialform sowie konstruktive Mitarbeit bei dieser Arbeit
- Klarheit und Richtigkeit beim Veranschaulichen, Zusammenfassen und Beschreiben informatischer Sachverhalte
- sichere Verfügbarkeit des Grundwissens der Informatik (z. B. Variablen Definition, Fallunterscheidung, Vererbung, Klassenbeziehungen, Schleifen, ... usw. .)
- situationsgerechtes Anwenden geübter Fertigkeiten
- angemessenes Verwenden der Fachsprache der Informatik
- konstruktives Umgehen mit Fehlern
- zielgerichtetes Beschaffen von Informationen
- Erstellen von nutzbaren Unterrichtsdokumentationen, ggf. Portfolio

- Klarheit, Strukturiertheit, Fokussierung, Zielbezogenheit und Adressatengerechtigkeit von Präsentationen, auch mediengestützt
- sachgerechte Kommunikationsfähigkeit in Unterrichtsgesprächen und Kleingruppenarbeiten
- Einbringen kreativer Ideen
- fachliche Richtigkeit bei kurzen, auf die Inhalte weniger vorangegangener Stunden beschränkten schriftlichen Überprüfungen

Die folgende Tabelle gibt eine Übersicht der Notenstufen zur Selbsteinschätzung der sonstigen Mitarbeit:

sehr gut	sehr kontinuierliche, ausgezeichnete Mitarbeit, sehr umfangreiche, produktive und kreative Beiträge, kommunikationsfördernd, souveräner Gebrauch der Fachsprache und souveräne Anwendung der Kenntnisse und Fähigkeiten, erscheint immer vorbereitet zum Unterricht.
gut	kontinuierliche, gute Mitarbeit, gute und produktive Beiträge, kommunikationsfördernd, sicherer Gebrauch der Fachsprache und sichere Anwendung Grundkenntnisse, erscheint meist vorbereitet zum Unterricht.
befriedigend	durchschnittliche Mitarbeit, kommunikativ, fachlich korrekte Beiträge, meistens sicherer Gebrauch der Fachsprache und sichere Anwendung der Grundkenntnisse, erscheint in der Regel vorbereitet zum Unterricht.
ausreichend	selten eigenständige Beteiligung, fachliche Ungenauigkeiten, auch unstrukturierte oder unproduktive Beiträge, kann sich grundlegend in der Fachsprache verständlich machen und Grundkenntnisse der Informatik in der Regel anwenden, erscheint häufig unvorbereitet zum.
mangelhaft	nur sporadische Mitarbeit trotz Aufforderung und Hilfsangeboten, schwerwiegende und anhaltende fachliche Defizite, meistens fehlerhafte oder lückenhafte Anwendung der Fachsprache und der Grundkenntnisse, erscheint trotz Aufforderung selten vorbereitet zum Unterricht.
ungenügend	keine Beteiligung trotz Aufforderung und Hilfsangeboten, fehlende fachliche Kenntnisse auch in elementaren Grundlagen, kann die Fachsprache nicht anwenden und sich mit ihr verständlich machen, es ist erkennbar, dass die Defizite nicht in absehbarer Zeit behoben werden können, erscheint unvorbereitet im Unterricht.

Nun aber endlich zur Informatik und natürlich viel Spaß beim Programmieren ;-)

## **Kapitel 2: Was ist Informatik, womit beschäftigt sich die Informatik**

Im Fachraum ist das Trinken und Essen nicht erlaubt. Der Arbeitsplatz ist grundsätzlich sauber zu hinterlassen. Dateien werden auf dem Laufwerk H (Netzwerklaufwerk) gespeichert und auf euren privaten USB-Stick. Es ist notwendig, dass ihr ein USB-Stick zu jeder Stunde mitbringt. Auch schafft ihr euch ein kariertes Heft an oder ihr legt eine Microsoft – Office – Word – bzw. LibreOffice – Writer – Datei an, in der ihr eure Lösungen der Aufgaben aufschreibt, aber wenn Zeichnungen erstellt werden sollen, dann wird es mit den Texteditoren etwas komplizierter. Für Klausuren sind mehrere karierte Bögen mitzubringen. Zum üben muss Zuhause ein Laptop oder ein PC vorhanden sein. Auf den eigenen Laptop oder PC muss die Software **Java** sowie **Java Editor** installiert sein.

### **Kapitel 2.1: Einige Begriffserläuterungen:**

Die **Informatik** ist die Wissenschaft von der Darstellung, Verarbeitung, Speicherung, Übertragung von Informationen in digitalen Rechnersystemen und beschäftigt sich mit dem Aufbau, Programmierung und Bedienung von Computern

Das Wort **Informatik** setzt sich aus den Wörtern Information und Automatik zusammen und bezeichnet die Wissenschaft von der systematischen Verarbeitung von Informationen mit Hilfe von Rechenanlagen (Rechnersystemen).

Die **Informatik** erfasst die Gesamtheit der Vorgänge, bei denen durch Verarbeitung von Daten über einen bestimmten Sachverhalt Informationen über den sich daraus ergebenden Sachverhalt gewonnen werden.

Da Rechenoperationen in Wissenschaft, Technik, Wirtschaft und Verwaltung in sehr großen Mengen durchzuführen sind, versuchte man die Informationsverarbeitung durch Rechenmaschinen zu rationalisieren. (Knaurs Lexikon a-z, 1969)

### **Kapitel 2.2: Teilgebiete der Informatik**

Man unterscheidet innerhalb der Informatik grob folgende Teilgebiete:

- I. Technische Informatik
- II. Praktische Informatik
- III. Theoretische Informatik
- IV. Angewandte Informatik

#### **Kapitel 2.2.1: Die Theoretische, Technische und Praktische Informatik.**

Die Theoretische, Technische und Praktische Informatik bilden zusammen die Informatik im engeren Sinne.

Man bezeichnet sie deshalb auch als Kerninformatik. Ihnen steht die Angewandte Informatik gegenüber, die die Anwendungsmöglichkeiten des Computers erforscht, sozusagen die Computer als Werkzeug zur Lösung von Aufgaben betrachtet.

Technische Informatik umfasst die ...

- ... Hardware („Eisenware“)
- ... physikalische Grundlagen, logische Schaltungstechnik
- ... technischer Aufbau von größeren Einheiten: arithmetisch logische Einheit (ALU), Speicher, Ein- und Ausgabegeräte
- ... Computerarchitektur, Computernetze
- ... Mikroprogrammierung

- ... maschinennahe Programmierung (Assembler)

Die Praktische Informatik umfasst die ...

- ... Prozesse der Software-Erstellung (*software engineering*)
- ... Höhere Programmiersprachen
- ... Betriebssysteme
- ... Datenbanken
- ... Mensch - Maschine – Kommunikation

Theoretische Informatik umfasst die ...

- ... Betrachtung von Computern als abstrakte Automaten
- ... Fragen nach prinzipiellen Grenzen von Computern und Algorithmen (Berechenbarkeit)
- ... Effizienz von Algorithmen (Komplexität)
- ... Formale Sprachen
- ... Künstliche Intelligenz, Expertensysteme

Angewandte Informatik die Bereiche ...

- ... Anwendung der Informatik auf spezielle Anwendungsgebiete (z.B. Wirtschaft)
- ... Anwendungsprogramme (Textverarbeitung, Tabellenkalkulation, Datenbank, Grafik)

### **Kapitel 2.3: Themengebiete in der Einführungsphase**

Die gesamte Einführungsphase beschäftigt sich mit dem Gebiet der Praktische Informatik und wir werden folgende Themen behandeln:

- Was ist Informatik? Teilgebiete der Informatik.
- Grundlagen der Programmierung. Programmiersprachen und Übersetzer. Der Compiler und Interpreter Java
- Grundlagen (mit Java Editor). Datentypen. Variablen und Konstanten. Operatoren. Kontrollstrukturen. Felder → Array
- Methoden in Java. Prinzip der Rekursion.
- Algorithmus. Definition, Eigenschaften und Darstellungsformen. Algorithmus und Programme.
- Suchen und Sortieren.
- Grafikprogrammierung (AWT, Swing).
- Klassen und Objekte → Grundlagen OOP
- UML Diagramme



## Kapitel 2.4: Computer und Programme; Algorithmus

Computer werden häufig zum Lösen von wiederkehrenden Problemen eingesetzt. Um dem Computer mitzuteilen, wie er ein bestimmtes Problem lösen soll, werden Programme geschrieben. Ein Programm beinhaltet eine Folge von Befehlen. Diese Befehle sind eine genau definierte Abfolge von Schritten, die zur Lösung eines Problems führen sollen. Man spricht dabei auch von einem Algorithmus.

Für das Ausführen des Algorithmus ist der Prozessor eines Computers verantwortlich. Der Prozessor liest die einzelnen Befehle und führt diese Schritt für Schritt aus. Damit der Prozessor die Befehle verstehen kann, müssen diese in Maschinencode vorliegen. Unter Windows haben Programme in Maschinencode häufig die Dateierweiterung „.exe“.

Maschinencode ist, einfach gesagt, eine hintereinander Reihung von Nummern (1-en und 0-en), wobei jede Nummer oder Nummernfolge für einen prozessorspezifischen Befehl steht.

## Kapitel 2.5: Programme lesen und ausführen

Die Grundbausteine eines jeden Computers ist der Prozessor, der Arbeitsspeicher und die Festplatte. Diese Bausteine spielen die wesentliche Rolle beim Ausführen und Lesen von Programmen. Programme werden z. B. auf der Festplatte eines Computers gespeichert und jedes Mal, wenn sie ausgeführt werden sollen, vom Betriebssystem in den Arbeitsspeicher des Computers geladen. Nachdem ein Programm im Arbeitsspeicher liegt, kann der Prozessor auf dieses zugreifen und alle Befehle, die in Maschinencode geschrieben wurden, mit Hilfe des Betriebssystems ausführen.

Unglücklicherweise versteht nicht jede Kombination aus Betriebssystem und Prozessor den gleichen Maschinencode. Aus diesem Grund müssen Programme meistens in verschiedenen Versionen, die unterschiedlichen Maschinencode beinhalten, vorliegen, damit sie auf jedem Computer ausführbar sind. Allerdings ist es nahezu unmöglich, komplexe Programme oder Programm-Versionen in Maschinencode zu schreiben. Aus diesem Grund wurden Programmiersprachen wie Java erfunden.

## Kapitel 2.6: Programmiersprachen

Programmiersprachen ermöglichen es, Programme in einer Sprache zu schreiben, die der Mensch einfacher erlernen und verstehen kann als Maschinencode. Die Befehle in einer Programmiersprache sind so definiert, dass man durch ihre Kombination wesentlich einfacher mitteilen kann, was der Computer bzw. Prozessor machen soll.

Wenn sich Befehle einer Programmiersprache in Form (Syntax) und Inhalt (Semantik) stark an der menschlichen Sprache orientieren, spricht man auch von einer Hochsprache. Java ist eine solche Hochsprache.

Ein Programm, das man in einer Hochsprache schreibt, muss man in einer Datei speichern. Der Inhalt einer solchen Datei wird Quellcode genannt. Allerdings lässt sich Quellcode nicht mehr ohne Hilfe ausführen, denn der Prozessor versteht ja nur Maschinencode. Aus diesem Grund muss ein Übersetzungsprogramm verwendet werden, das aus Quellcode Maschinencode macht. Bei den Übersetzungsprogrammen kann zwischen Compiler und Interpreter unterschieden werden.

## Kapitel 2.7: Der Compiler

Die Aufgabe des Compilers ist es, Quellcode in Maschinencode zu übersetzen. Der Compiler liest den Quellcode und erzeugt ein ausführbares Programm in Maschinencode. Dieses Programm wird auf der Festplatte oder einem anderen Datenträger gespeichert und kann beliebig oft durch einen Doppelklick oder einen anderen Befehl ausgeführt werden. Die Funktion des Compilers ist vergleichbar mit der eines Übersetzers, der ein englisches Buch liest, übersetzt und eine neue Ausgabe in Deutsch erzeugt.

## Kapitel 2.8: Der Interpreter

Neben der Methode des Übersetzens durch einen Compiler gibt es noch eine Alternative - den Interpreter. Ein Interpreter übersetzt, wie der Compiler, Quellcode. Anstatt ein Programm zu erstellen, führt der Interpreter jede Anweisung, die er übersetzt, direkt aus. Im Vergleich zu einem klassischen Übersetzer ist der Interpreter ein Simultan-Dolmetscher, der alles was er liest direkt übersetzt und ausführt ohne ein neues Dokument zu erzeugen.

## Kapitel 2.9: Compiler vs. Interpreter

Sowohl der Compiler als auch der Interpreter erfüllen die gleiche Aufgabe - beide übersetzen Quellcode. Obwohl beide Übersetzerprogramme im Ergebnis das Gleiche tun, hat sowohl der Compiler als auch der Interpreter seine Vor- und Nachteile. Diese werden im Folgenden kurz aufgeführt:

- **Kompilierte Programme sind schneller:** Programme, die mit einem Compiler erzeugt werden, lassen sich schneller ausführen als interpretierte Programme. Ein kompiliertes Programm liegt in Maschinencode vor und muss lediglich vom Prozessor gelesen werden.
- **Kompilierte Programme sind schwer zu pflegen:** Wird der Quellcode eines kompilierten Programms verändert, geschieht zunächst nichts. Solange der Quellcode nicht zu einem neuen Programm kompiliert wurde, steht die Änderung nicht im Maschinencode und der Prozessor kann sie auch nicht ausführen.
- **Interpretierte Programme sind pflegeleicht:** Der Quellcode eines interpretierten Programms wird jedes Mal zur Laufzeit neu übersetzt. Das bedeutet, jede Änderung am Quellcode wird direkt in Maschinencode umgewandelt und ausgeführt.
- **Interpretierte Programme sind langsamer:** Da ein interpretiertes Programm zur Laufzeit übersetzt und dann ausgeführt wird, sind diese langsamer als kompilierte Programme. Ein kompiliertes Programm liegt bereits in Maschinencode vor und muss lediglich ausgeführt werden.

## Kapitel 2.10: Allgemeines über Java

Java hat zudem eine Besonderheit im Vergleich zu klassischen Hochsprachen: Ein in Java geschriebenes Programm ist in seinem fertigen Zustand unabhängig von vielen Plattformen. Das heißt, ein Java-Programm kann auf fast jeder Kombination von Betriebssystem und Prozessor ausgeführt werden. Um diesen Zustand zu erreichen, wird der Quellcode eines Java-Programms aus einer Kombination von Compiler und Interpreter (in Java heißt er "Java Virtual Machine" (JVM)) übersetzt. Der Java-Compiler erzeugt aus Java-Quellcode keinen prozessorspezifischen Maschinencode, sondern sogenannten Bytecode (Dateien, in denen Bytecode steht, haben die Endung „.class“). Bytecode ist ein unabhängiger Maschinencode, der nicht direkt von einem Prozessor ausgeführt werden kann. Stattdessen muss der Bytecode eines Java-Programms von der Java Virtual Machine (JVM) eingelesen und ausgeführt werden. Die JVM ist der Interpreter in Java und Bestandteil jeder Java-Installation. Da Java für die gängigsten Betriebssystem-Prozessor-Kombinationen bereitgestellt wird, kann auch Java-Bytecode auf den meisten Computern ausgeführt werden.

## Kapitel 2.11: Die Vorteile von Java-Bytecode

Vorteile, die aus der Kombination von Java-Compiler und Java Virtual Machine entstehen:

- **Geschwindigkeit:** Der Java-Compiler bereitet den Java-Bytecode so vor, dass er wesentlich schneller von der JVM interpretiert werden kann als normaler Quellcode.
- **Einmal schreiben, überall ausführen:** Ein kompiliertes Java-Programm kann auf jedem Computer ausgeführt werden, wenn die Java Virtual Machine installiert ist. Da für die gängigsten Betriebssystem-Prozessor-Kombinationen die JVM zur Verfügung steht, können kompilierte Java-Programme (also Java-Bytecode) auf fast allen Computern ausgeführt werden. Mehrere Programm-Versionen für unterschiedliche Betriebssysteme und Prozessoren sind nicht nötig (diese Eigenschaft wird *Plattformunabhängigkeit* genannt)

## Kapitel 3: Erste Schritte mit Java

### Kapitel 3.1: Vorbereitung zum Programmieren

Bevor Sie in die Programmierung Ihres ersten Java-Programms einsteigen, müssen Sie zwei Entwicklungswerkzeuge installieren. Das Java Development Kit (JDK) und eine Programmierumgebung. Das JDK ist das Herzstück einer Java-Installation und die Programmierumgebung wird zum Schreiben des Quellcodes benötigt.

### Kapitel 3.2: Das Java Development Kit (JDK)

Das Java Development Kit (JDK) beinhaltet den Java-Compiler und die Java Virtual Machine (JVM). Es wird von der Firma Sun Microsystems kostenlos zur Verfügung gestellt, Wie Sie das JDK auf auf euren PC oder Laptop installieren zeigen Ihnen die folgenden Links:

<https://techexpert.tips/de/windows-de/java-jdk-installation-unter-windows/>

oder ein Video „wie man das JDK installiert“

<https://www.youtube.com/watch?v=U1KQrPl3FPg>

und noch ein Video „wie man das JDK installiert“

<https://www.youtube.com/watch?v=HuYs2Vp3GtM> ab der Stelle 08:29

Für diejenigen von Ihnen, die sich mit der Installation von Programmen besser auskennen folgt eine kurze und knappe Anleitung.

Auf dieser Website (<https://techexpert.tips/de/windows-de/java-jdk-installation-unter-windows/>) stehen verschiedene Versionen des JDK zum Download bereit (z. B. Version JDK 18.xx.xx). **Es muss aber unbedingt das JDK und nicht nur die JRE sein.** Wenn Sie auf den Download-Button klicken, müssen Sie Ihr Betriebssystem auswählen und auf „Continue“ klicken. Im Folgebildschirm klicken Sie auf die angebotene Datei. Die Installation wird durch Aufrufen der heruntergeladenen Datei gestartet. Dabei kann einfach den Anweisungen gefolgt werden.

### Kapitel 3.3: Die Programmierumgebung → Installation des Java-Editors (Zu Hause)

Die aktuelle Version des Java-Editors können Sie – selbstverständlich kostenlos – über den Link <http://javaeditor.org/doku.php?id=en:download> herunterladen. Da Sie das Programm zu Hause auf einem einzelnen PC installieren, sollten Sie die „Personal Version“ wählen und danach die aktuelle Version.

Bei der Installation erkennt der Java-Editor automatisch das installierte JDK. Im Anschluss sind noch weitere Konfigurationseinstellungen für den Java-Editor erforderlich bzw. sinnvoll.

Installation von Java und des Editors "javaeditor": <https://www.youtube.com/watch?v=xyxTovQubGo>

### Kapitel 3.3.1: Systemvariablen anpassen (Zu Hause)

Die Installation des JDK haben Sie abgeschlossen. Jetzt müssen Sie die Systemvariablen anpassen. Führen Sie die folgenden Schritte durch:

Suchen Sie über das Icon „Lupe“ nach „umgebungsvariablen“  
Es erscheint das ein Fenster wie in Abbildung 2

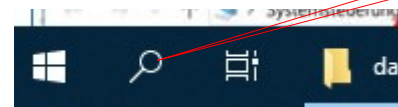


Abbildung 1

Klicken Sie auf den Button „umgebungsvariablen“. Es öffnet sich ein weiteres Fenster, wie in Abbildung 3.

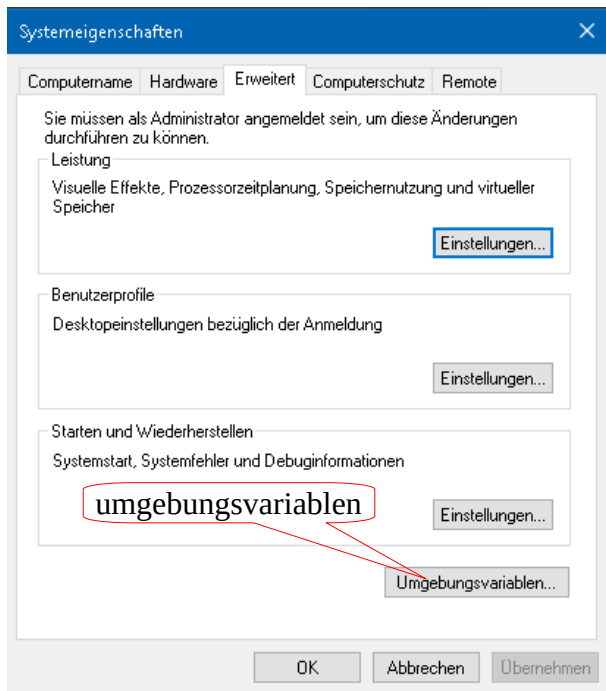


Abbildung 2

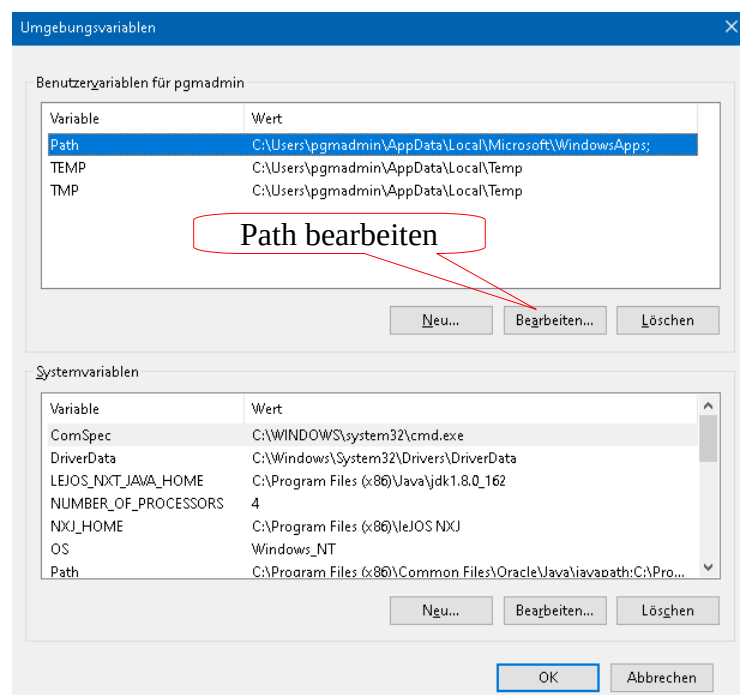


Abbildung 3

Sie müssen nur noch die Variable „path“ um ein Verzeichnis ergänzen. Klicken Sie auf „bearbeiten“.

Es öffnet sich ein weiteres Fenster und dort klicken Sie auf „Text bearbeiten“. Und es öffnet sich noch ein Fenster :-). (Siehe Abbildung 5). Dort in diesem Fenster tragen Sie nach dem Semikolon „;“ (mit der Mouse setzen Sie den Cursor nach dem Semikolon) das Installationsverzeichnis der JDK-Installation. Das Installationsverzeichnis können Sie über „verzeichnis durchsuchen“ auswählen. Meistens befindet sich die JDK-Installation im Verzeichnis „C:/Programme/java/jdk13.xx.xx/bin“.

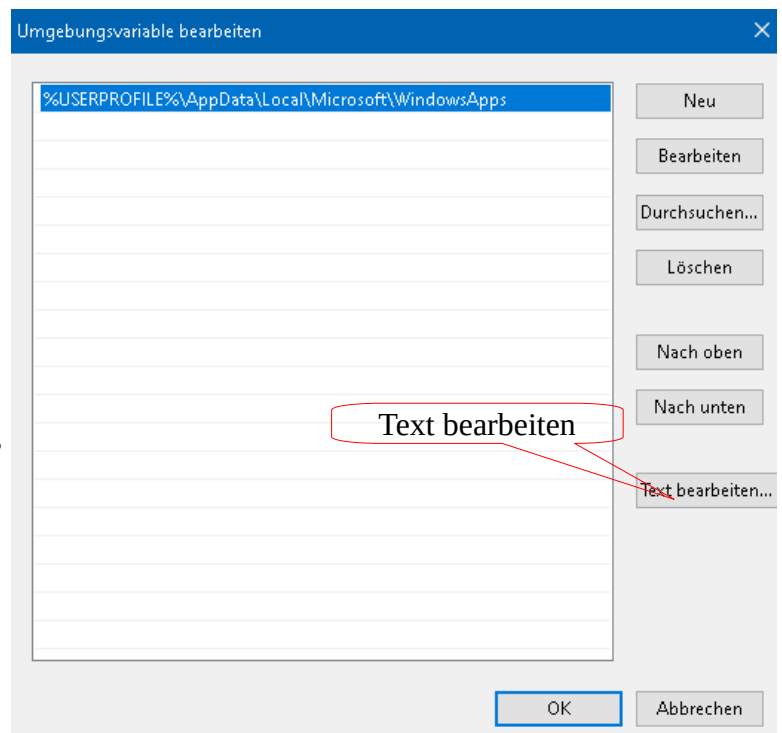


Abbildung 4

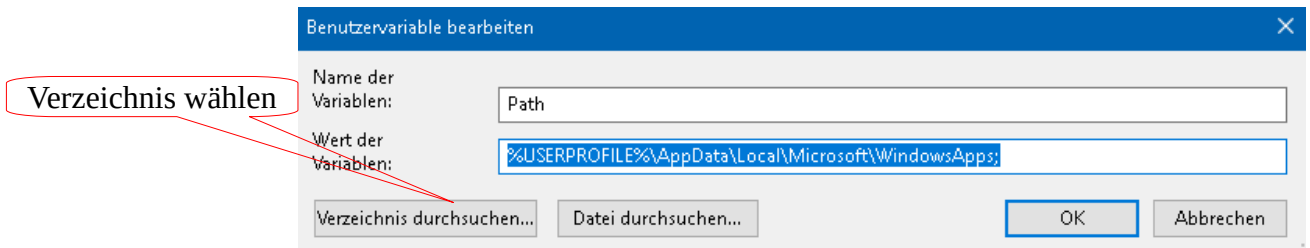


Abbildung 5

Hier ein Video dazu: <https://www.youtube.com/watch?v=GKIbXQ2kt7o> ab 1:49.

Oder die Installation von JDK und Javaeditor:  
<https://www.youtube.com/watch?v=OB8mNiFZpSs>

### Kapitel 3.4: Die Alternative Möglichkeit die Systemvariablen über den „JavaEditor“ anpassen (Zu Hause)

Unter dem Menüpunkt „Fenster → Konfiguration“ finden Sie das Fenster, in dem Sie angeben können wo das JDK installiert ist (Siehe Abbildung 6).

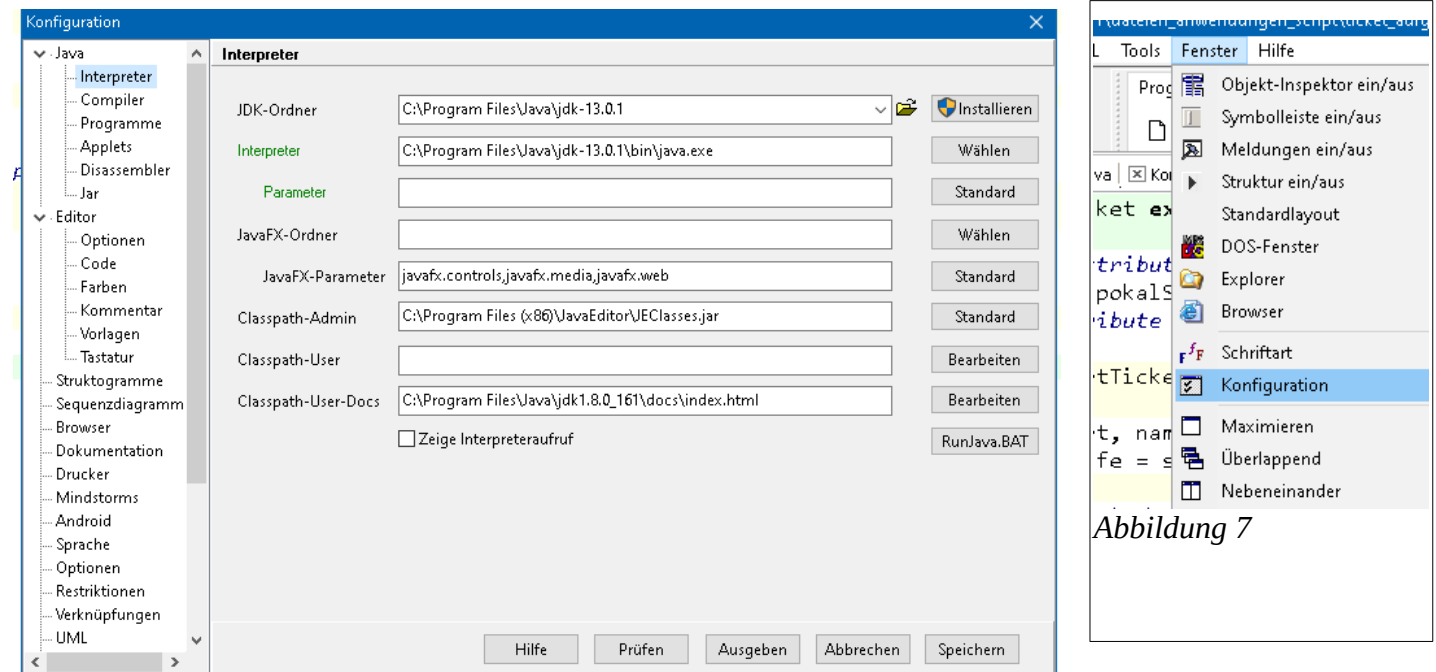


Abbildung 6

Die Installation von JDK und Javaeditor:  
<https://www.youtube.com/watch?v=OB8mNiFZpSs>

### Kapitel 3.5: Die Ordnerstruktur anlegen (In der Schule)

Nachdem das JDK und der Programmierungsumgebung eingerichtet wurden, müssen Sie eine Ordnerstruktur anlegen, denn Java-Programme sollten nicht einfach willkürlich auf der Festplatte des Computers abgespeichert werden. Legen Sie sich bitte ein Projektverzeichnis „[H:/Informatik\\_EF](#)“ (das Verzeichnis trägt Ihren anmeldekürzel!!!) an. Erstellen Sie in „Informatik\_EF“ für jedes Arbeitsblatt einen neuen Ordner, in dem Sie dann Ihre Java-Programme erstellen.

### Kapitel 3.6: Den ersten Java-Quellcode erstellen , speichern und ausführen

Jetzt kann es mit der Programmierung des ersten Java-Programms losgehen. Dafür müssen wir mit der Erstellung des Java-Quellcodes beginnen. Das erste Java-Programm soll den Nutzer mit dem Satz "Hallo, Java!" begrüßen. Dafür muss der folgende Quellcode geschrieben werden:

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

Abbildung 8

Starten Sie den JavaEditor (Menü → JavaEditor →  JavaEditor ), erstellen Sie eine neue Java-Datei (Datei → Neu → Java) und geben Sie den Quellcode ein. Speichern Sie den Quellcode in der Datei „HalloJava.java“ (Datei → Speichern unter...). Wählen Sie als Speicherort das Verzeichnis [H:/Informatik\\_EF/Arbeitsblatt\\_1](#).

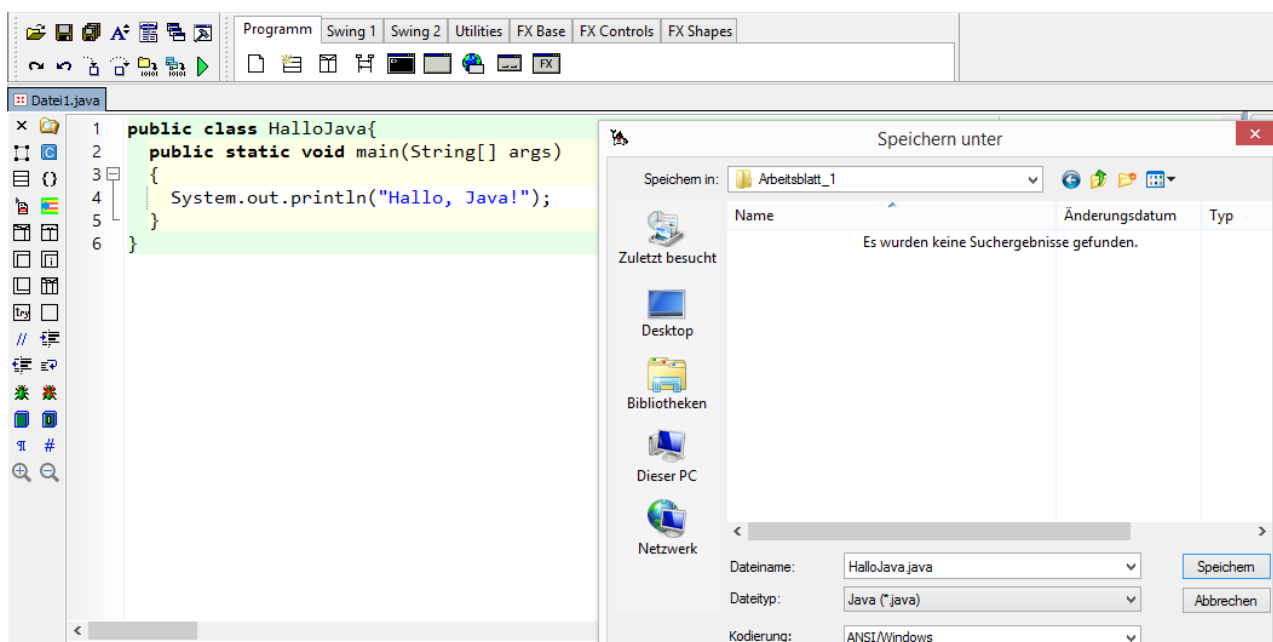




Abbildung 9: „HalloWelt.java“ speichern

Beim Starten des Programms (Start → Starte Applikation  oder F9 oder ) wird der Java-Compiler aufgerufen und den Java-Quellcode in Bytecode umwandeln und eine class-Datei erstellen. Im Verzeichnis „[H:/Informatik\\_EF/Arbeitsblatt\\_1](#)“ sollte jetzt „HalloJava.class“ gespeichert sein. Die class-Datei „HalloJava.class“ ist ein ausführbares Java-Programm in Bytecode.

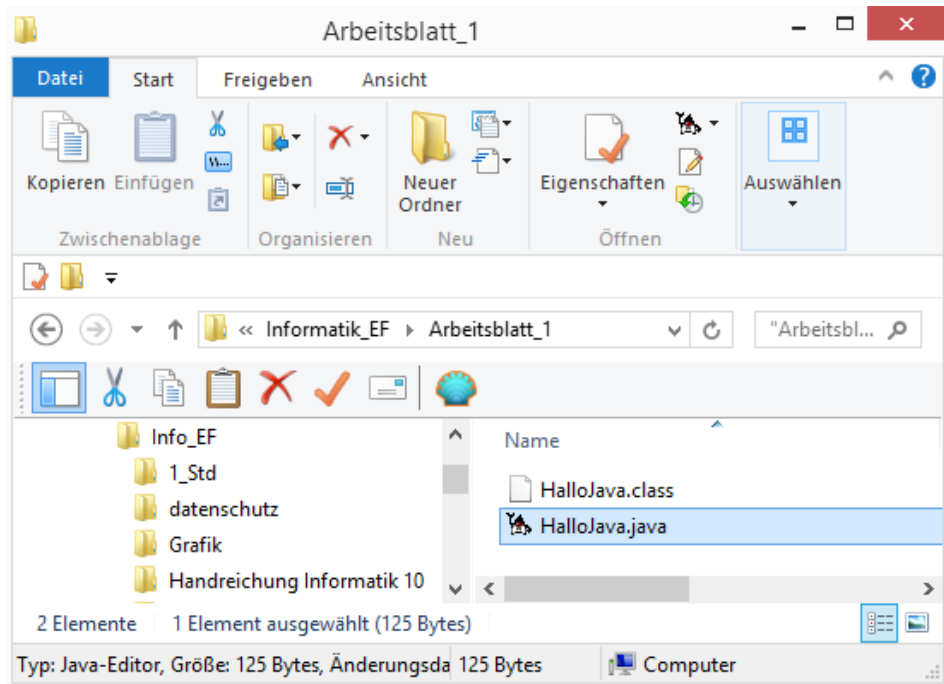


Abbildung 10 „HalloJava.class“ erstellen

Nach dem Starten des Programms, sollte auf der Konsole der Satz "Hallo, Java!" erscheinen. Erscheint der Satz, haben Sie es geschafft. Sie haben ihr erstes Java-Programm geschrieben, kompiliert und ausgeführt ;-)

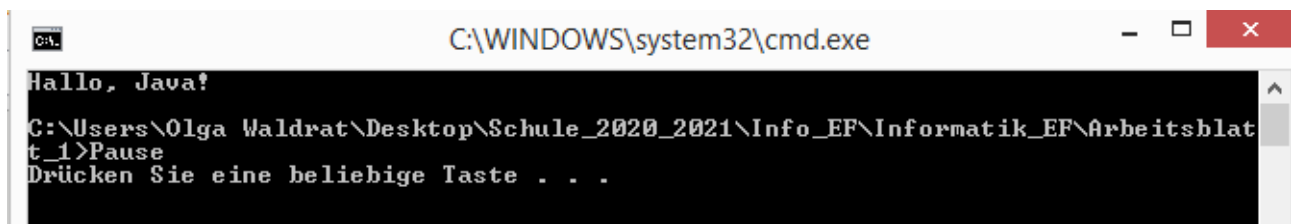


Abbildung 11 Konsolenausgabe

**Aufgabe 1:** Erweitere die Klasse „HalloJava“ so, dass die folgenden Ausgaben auf die Konsole ausgegeben werden:

- „Das ist mein erstes Programm in Java“
- „Geben Sie ein Wert für die Variable a ein:“
- „Geben Sie ein Wert für die Variable b ein:“

**Aufgabe 2:** Texte werden in JAVA mit einem plus „+“ zusammengefügt. Fügen Sie die jeweils getrennten Wörter mit einem Pluszeichen zusammen (Z.B: System.out.println("Groß " + " und klein"); ):

- „Hallo“ und „Welt“
- „Der “ und „ Ball “ und „ hüpf “
- „Lalala “ und „ lululu “ und „ lilili “



## Kapitel 3.7: Strukturelemente und Anweisungen

### Kapitel 3.7.1: Die Rolle der Klasse

Wenn man ein Java-Programm entwickeln möchte, muss man den Quellcode für dieses Programm in eine Java-Datei (z. B. HalloJava.java) schreiben. Diese Datei kann dann kompiliert und ausgeführt werden. Beim Schreiben des Quellcodes muss man immer einer fest vorgegebenen Struktur folgen:

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

Ganz am Anfang des Quellcodes muss eine **Klassendefinition** stehen. In der **Klasse** können Methoden angelegt werden, die Anweisungen beinhalten. Damit die Java-Datei als Programm ausgeführt werden kann, muss mindestens die sogenannte **main**-Methode in der Klasse vorhanden sein.

#### Kapitel 3.7.1.1: Die HalloJava-Klasse

Der Java-Quellcode in einer Java-Datei muss immer mit einer **Klassendefinition** eingeleitet werden. Das ist sehr **wichtig**, dass Sie sich den Aufbau einer Klasse merken:

```
public class HalloJava
{
    ...
}
```

Abbildung 12

Die Klassendefinition des vorliegenden Java-Programms beginnt in der ersten Zeile und wird mit den Wörtern **public** und **class** eingeleitet. Anschließend folgt der **Name der Klasse**. Hier heißt die **Klasse HalloJava**. Wichtig ist, dass eine Klasse immer so benannt sein muss, wie die Java-Datei, in der die Klasse definiert wird.

Der Beginn und das Ende einer Klassendefinition wird mit einer geschweiften Klammer „{ }“ gekennzeichnet. Innerhalb der geschweiften Klammern können Methoden und Anweisungen formuliert werden.

#### Kapitel 3.7.1.2: Aufbau einer Methode

Methoden sind Strukturierungselemente in Java, die als Behälter für Anweisungen eingesetzt werden. Alle Anweisungen, die eine Klasse bereitstellen soll, müssen in Methoden definiert sein. Eine Methode besteht immer aus einem Methodenkopf und einem Methodenkörper und muss in einer Klasse angelegt werden. Das heißt, sie muss zwischen den geschweiften Klammern der Klassendefinition stehen:

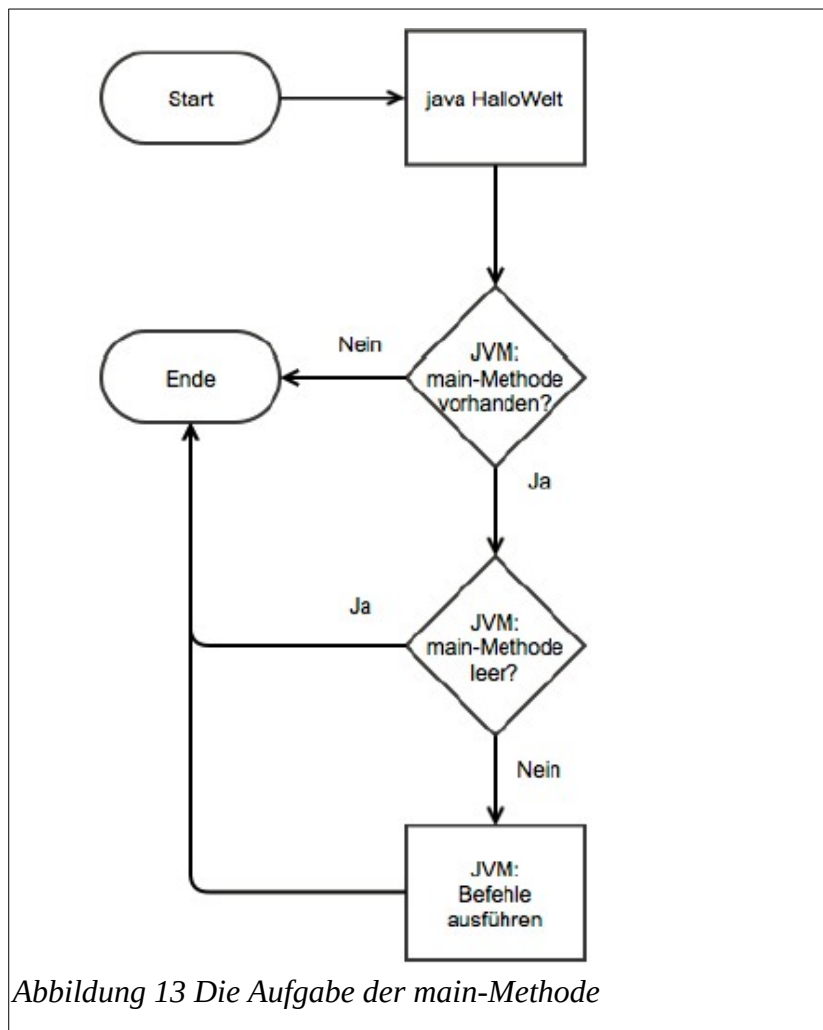
```
public class HalloJava
{
    public static void main(String[] args)
    {
    }
}
```

Ein **Methodenkopf** enthält die Worte **public** und **static**, den **Namen der Methode** und **Parameter**. Hier heißt die Methode **main** und hat **String[] args** als Parameter. Ein Methodenname wird immer mit einem **kleinen Buchstaben** eingeleitet. Der Methodenkörper kann Befehle enthalten, die von der Methode ausgeführt werden sollen. Beginn und Ende des Methodenkörpers werden durch geschweifte Klammern { } gekennzeichnet.

Die Klasse HalloJava beinhaltet die Methode mit dem Namen „**main**“. Diese Methode erfüllt eine besondere Rolle in Java-Programmen.

### Kapitel 3.7.1.3: Die Rolle der main-Methode

Eine Java-Datei und die darin enthaltene Klasse kann nur als eigenständiges Programm ausgeführt werden, wenn sie eine **main**-Methode beinhaltet.



Eine **main**-Methode wird immer mit „**public static void main (String[] args)**“ deklariert. Anschließend folgt der Methodenkörper. Im Methodenkörper der **main**-Methode muss nacheinander beschrieben werden, was das Programm machen soll. Um das zu tun, verwendet man Anweisungen. Die **main**-Methode der vorliegenden HalloJava-Klasse beinhaltet eine Anweisung

```
System.out.println("Hallo, Java!");
```

An dieser Stelle ist es wichtig, dass Sie sich den Aufbau der **main**-Methode genau merken.

Ein Video dazu: Erste Schritte mit dem JAVA-Editor: <https://www.youtube.com/watch?v=-79pYNUaua4>

### Kapitel 3.7.1.4: Bedeutung der Schlüsselwörter:

- **public**: Bezeichner (Modifikator) für die Zugriffsart, bedeutet dass eventuell noch weitere definierte Klassen auf diese Methoden zugreifen können (d. b. sie benutzen dürfen)
- **static**: Der Modifikator zeigt, dass es sich hier um eine sogenannte Klassenmethode handelt.
- **void**: Bezeichnet der Typ der Rückgabeveriable. Die Rückgabeveriable ist das, was eine Methode ausrechnet. Die **main**-Methode selbst ausrechnet nichts, deswegen steht an dieser Stelle immer **void** (eng. „leer“).

- **main:** Name der Methode (hier Hauptprogramm).
- **(String [] args):** in den Klammern nach dem Methodennamen stehen die Typen und die Namen der Variablen, mit denen die Methode arbeitet (die sogenannten Argumente). Bei der **main-Methode** ist das Argument immer „String [] args“ oder „String args []“. Dabei ist String der Variablentyp (eine Zeichenkette), args der Variablenname (Abkürzung für „arguments“), die eckigen Klammern [] zeigen an, dass die Argumente eine Reihung von Strings (ein sogenanntes Array) sind.

### Kapitel 3.7.1.5: Die Ausgabeanweisung

Mit der Anweisung **System.out.println();** kann man Sätze auf dem Bildschirm - genauer gesagt in der Konsole anzeigen. In der Klammer von **println()** muss in Anführungszeichen stehen, was angezeigt werden soll.

Der Nutzer des vorliegenden Programms sollte mit "**Hallo, Java!**" begrüßt werden, deshalb wurde „**System.out.println("Hallo, Java!");**“ in den Methodenkörper der **main**-Methode geschrieben. Diese Anweisung kann man auch beliebig oft wiederholen und nacheinander unterschiedliche Sätze oder Worte auf dem Bildschirm anzeigen lassen:

```
public class HalloWelt
{
    public static void main(String[] args)
    {
        System.out.println("Hallo, Java!");
        System.out.println("Mein Name ist");
        System.out.println("Max Mustermann");
    }
}
```

Abbildung 14: Erweitertes HalloJava-Programm

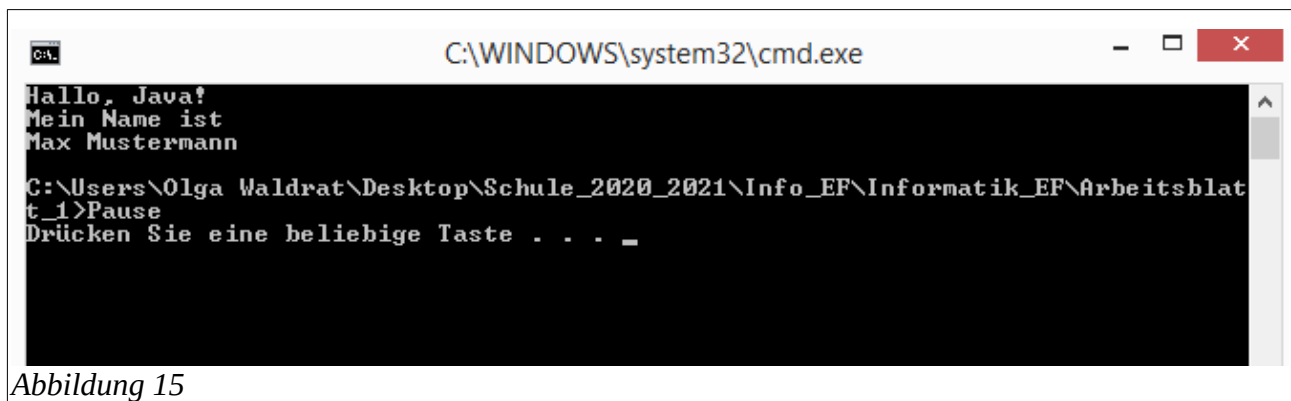


Abbildung 15

### Kapitel 3.7.1.6: Kommentieren und Einrücken

Nachdem man Neues gelernt hat, ist es sinnvoll sich Notizen zu machen. Das geht nicht nur auf Papier, sondern auch im Quellcode. Notizen im Quellcode werden Kommentare genannt. Kommentare können in zwei verschiedenen Arten in Java-Quellcode eingebaut werden. Man kann einzeilige Kommentare oder Kommentarblöcke anlegen:

```
/* Das ist ein mehrzeiliger Kommentar für die Klasse HalloJava. Hallo-Java liegt in der Datei
HalloJava.java und beinhaltet eine main-Methode und Ausgabe-Aeusungen. */
public class HalloJava
{
    //main-Methode
    public static void main(String[] args)
    {
        //Anweisungen
        System.out.println("Hallo, Java!");
        System.out.println("Mein Name ist");
        System.out.println("Max Mustermann");
    }
}
```

Abbildung 16

Ein einzeiliger Kommentar wird mit „`/*`“ eingeleitet. Alles was dahinter steht, wird vom Compiler ignoriert. In der nächsten Zeile geht es aber ganz normal weiter. Möchte man länger Kommentare schreiben, kann man einen Kommentarblock erstellen. Ein Kommentarblock wird mit „`/*`“ geöffnet und mit „`*/`“ geschlossen.

Eine weitere Maßnahme zur übersichtlichen Programmierung ist das **Einrücken** von Quellcode. Die Entwickler von Java schreiben in sogenannten Quellcode-Konventionen vor, dass man alles, was innerhalb von geschweiften Klammern „`{ }`“ passiert, einrücken soll. Gerade am Anfang kommt es häufig vor, dass Klammern vergessen werden. Durch konsequentes Einrücken sind solche Fehler gut sichtbar und außerdem erleichtern die verschiedenen Einrücktiefen das Lesen des Quellcodes wesentlich.

**Aufgabe 3:** Erklären Sie, wie eine Methode und eine Klasse aufgebaut ist.

**Aufgabe 4:** Erklären Sie, welche Funktion Methoden in der Programmiersprache haben (Wofür braucht man die?!).

**Aufgabe 5:** Erklären Sie, welche Bedeutung die Schlüsselwörter „`static`“, „`public`“ und „`void`“ besitzen.

**Aufgabe 6:** Geben Sie die Rolle der `main`-Methode an.

**Aufgabe 7:** Erklären Sie die Bedeutung der eckigen Klammern im Methodenkopf.

**Aufgabe 8:** Erklären Sie, was ein Kommentar ist und welche Arten von Kommentaren es gibt.

**Aufgabe 9:** Erklären Sie, was die Unterschiede zwischen einem Quellcode und einem Kommentar sind.

**Aufgabe 10:** Im folgenden Quelltext sind etliche Fehler. Finden Sie diese und begründen Sie Ihre Entscheidung. Korrigieren Sie die Fehler. Ihre Begründung fügen Sie mit Kommentarzeichen in den Quelltext ein.

```
/* Das ist ein mehrzeiliger Kommentar für die Klasse HalloJava. Hallo-Java liegt in der Datei
HalloJava.java und beinhaltet eine main-Methode und Ausgabe-Aeusungen.
```

```
public clas HalloJava
{
    /main-Methode
    public void staic main(String[ args)

        //Anweisungen
        System.out.println("Hallo, Java!");
        System.out.println("Mein Name ist")
        System.out.println("Max Mustermann" + " Klaudius Romanum );
}
```

Abbildung 17

## Kapitel 3.7.2: Variablen und Konstanten

### Kapitel 3.7.2.1: Die Rolle der Variablen

Damit man mit einer Klasse nicht nur Texte ausgeben kann, sondern auch komplexere Problemstellungen bearbeiten kann, muss man Variablen anlegen. Die **Variablen** dienen als Platzhalter für die Werte im Speicher und belegen je nach Datentyp mehr oder weniger Ressourcen. In den Variablen werden Zahlen oder Buchstaben abgespeichert. In Java werden Variablen z. B. in Methoden einer Klasse angelegt:

```
public class Bsp_prog1 {
    public static void main(String[] args) {
        int a,b,c; // Variablendeklaration
        // Wertzuweisung → Initialisierung
        a=6;
        b=7;
        c=a*b;
        System.out.println(c); // Ausgabe
    }
}
```

Abbildung 18

Das Programm besteht aus folgenden Komponenten:

- einer Klasse mit dem Namen **Bsp\_prog1**
- einer Methode mit dem Namen **main ()**
- drei Variablen vom Typ **int**
- drei Wertzuweisungen (**a=6; b=7; c=a\*b;**)
- einer Anweisung (**System.out.println**)
- mehreren Schlüsselwörtern (**public, int, class** usw.)
- Kommentaren (nach **//** oder zwischen **/\*** und **\*/**)

Diese und weitere Komponenten sind in fast jedem Java-Programm enthalten.

Auch hierzu gibt es Videos:

Variablen: [https://www.youtube.com/watch?v=sJ5Y7-R1e\\_U&list=PLbQQrHdDAuHAIpBxRFxnKv6WU8PAZmKHy&index=3](https://www.youtube.com/watch?v=sJ5Y7-R1e_U&list=PLbQQrHdDAuHAIpBxRFxnKv6WU8PAZmKHy&index=3)

### Kapitel 3.7.2.2: Variablen anlegen

Variablen werden mit einer Variablendefinition angelegt und mit Werten bestückt. Eine Variablendefinition beginnt mit einem Datentyp, gefolgt vom Namen der Variablen. Wenn man den Datentyp und Namen in den Quellcode geschrieben hat, spricht man von einer **Deklaration**. Ab jetzt weiß Java, dass es die Variable gibt:

```
//Datei Mensch.java
public class Mensch {
    public static void main(String[] args) {
        String name = „Max“; //Deklaration & Initialisierung
        int alter; //Deklaration
        alter = 35; //Initialisierung
    }
}
```

Abbildung 19

In der vorliegenden Klasse **Mensch** wurden die Variablen „**name**“ und „**alter**“ mit dem Datentyp „**String**“ und „**int**“ deklariert.

Der Name einer Variablen kann nach folgenden Regeln gebildet werden:

- der Name beginnt mit einem kleinen Buchstaben.
- Anschließend folgt eine beliebige Folge von Buchstaben, Zahlen und zulässigen Zeichen.
- der Unterstrich kann wie ein Buchstabe eingesetzt werden.
- Groß- und Kleinschreibung wird unterschieden.
- Operatoren und Schlüsselwörter sind nicht erlaubt.

Damit die Variablen eine sinnvolle Aufgabe im Programm erfüllen können, wird ihnen ein Wert zugewiesen. Die erstmalige Zuweisung eines Wertes nennt man **Initialisierung**. Die Initialisierung erfolgt durch ein Gleichzeichen und kann auf zwei Arten erfolgen: Entweder direkt in der Zeile, in der die Variable bekanntgegeben wurde oder in einem späteren Abschnitt des Quellcodes.

### Kapitel 3.7.2.3: Einfache Datentypen

Jedes mal, wenn Sie eine Variable anlegen, müssen Sie mit einem Datentyp bestimmen, welcher Wert in der Variablen abgelegt werden soll. Dafür wurde in der Klasse zuvor **String** und **int** benutzt. Neben dem Datentypen **String** und **int** gibt es noch andere Datentypen, die für Variablen genutzt werden können.

Mit diesen sogenannten *einfachen Datentypen* kann man ganze Zahlen, Fließkommazahlen oder einzelne Zeichen in einer Variablen speichern. Wenn man eine Zahl speichern will, kann man zwischen **byte**, **short**, **int**, **long**, **float** oder **double** wählen:

Datentyp	Wertebereich	Wertetyp
boolean	True , False	Wahrheitswert
char	90.000 Zeichen (Unicode)	Buchstaben und Zeichen
byte	-128 bis +127	Ganze Zahl
short	-32.768 bis +32.767	Ganze Zahl
int	-2.147.483.648 bis +2.147.483.647	Ganze Zahl
long	-9.223.372.036.854.775.808 bis +9.223.372.036.854.775.807	Ganze Zahl
float	+/- 10 <sup>38</sup>	Fließkommazahl
double	+/- 10 <sup>308</sup>	Fließkommazahl

Abbildung 21 Einfache Datentypen

```
//Beispiele
boolean wahr = true;
char zeichen = 'a';
byte alter = 100;
short jahr = 2015;
int nummer = 82312;
long bevoelkerung = 720000;
float temperatur = 30.90f;
double PI = 3.14159265359;
```

Abbildung 20

Variablen vom Typ **int** können z. B. ganze Zahlen, die größer als -2.147.483.648 und kleiner als +2.147.483.648 sind, speichern. **double** und **float** erlauben dagegen das Abspeichern wesentlich größerer Zahlen, inklusive Komma. Wenn Sie Kommazahlen in einer Variable vom Typ float speichern wollen, müssen Sie immer ein „f“ hinter den Wert schreiben (z. B. **float temperatur = 30.9f**).

#### Kapitel 3.7.2.4: Der Datentyp char

Zeichen oder Symbole werden durch den Datentyp *char* identifiziert und zwischen folgender Zeichen `` geschrieben. Beispiel:

```
char d=`7`;    // das Zeichen 7
char e=`b`;    // der Buchstabe b
```

#### Kapitel 3.7.2.5: Der Datentyp boolean

Der *boolean* bezeichnet einen Wahrheitswert und kann demzufolge *true* (wahr) oder *false* (falsch) sein. Beispiel

```
boolean b;
b = false;
```

Bei Fallunterscheidungen werden sehr oft boolean Variablen verwendet. Das Thema „Fallunterscheidungen“ wird in einem späteren Kapitel behandelt.

#### Kapitel 3.7.2.6: Der Datentyp String

Ein besonderer Datentyp ist der Datentyp **String**. Variablen, die mit dem Datentyp String angelegt werden, können Zeichenketten speichern. Zeichenketten werden durch die Klasse *String* dargestellt. Wird einer Variablen vom Typ *String* ein Wert zugewiesen, muss dieser in doppelten Anführungszeichen stehen. Eine Zeichenkette kann ein einfaches Zeichen sein, mehrere Zeichen, Wörter oder ganze Sätze. Um eine Variable mit dem Datentyp String zu deklarieren, schreiben Sie erst den Datentyp String hin und dann folgt der Name:

**String zeichenkette = "Hallo, Java!";**

Ein String wird, wie einfachen Datentypen, mit einem Gleichzeichen initialisiert. Zeichenketten, die in einem String gespeichert werden sollen, müssen immer in Anführungszeichen stehen.

Beispiel:

```
class Test_String
{
    public static void main(String args[])
    {
        String name = "Harry Potter";
        String beruf = "\"Zauberer\"";
        System.out.println(name + " ist ein " + beruf);
    }
}
Abbildung 22
```

Ergebnis:

Harry Potter ist ein 'Zauberer'

### Kapitel 3.7.2.7: Umwandlungen von Datentypen.

Es ist oft notwendig, den Wert eines primitiven Datentyps zu kopieren. Dabei können die Werte auch zwischen unterschiedlichen primitiven Datentypen ausgetauscht werden. Bei zwei gleichen Datentypen ist das kein Problem. Aber, wenn wir den Wert, der in einem Datentyp A gespeichert ist, an einen anderen Datentyp B übergeben wollen, dann können zwei Fälle auftreten nämlich:

Datentyp A in den größeren Datentyp B kopieren. Funktioniert ohne Problem, weil in dem größeren Datentyp mehr Speicher zur Verfügung steht.

Datentyp B in den kleineren Datentyp A kopieren. Dabei kann es zu Datenverlust kommen und in jedem Fall wird der Java-Compiler eine Fehlermeldung ausgeben.

### Kapitel 3.7.2.8: Explizite Typumwandlung.

Es gibt eine Möglichkeit, den Kopiervorgang des Inhaltes des Datentyps in einen kleineren zu erzwingen. Das nennt man explizite Typumwandlung oder „Casten“. Man sollte das nur dann benutzen, wenn man weißt, dass Daten verloren gehen können und es in diesem Fall entweder unwichtig ist, oder der Inhalt definitiv passt. Dazu schreibt man in Klammern vor die Zuweisung den neuen Datentyp. Beispiel:

```
float f;  
double d = 2.3;  
f = (float) d;
```

### Kapitel 3.7.2.9: Übersicht zu impliziten Typumwandlungen

Es gelten für die Zahlentypen folgende Größenverhältnisse:

```
byte < short < int < long  
float < double
```

Beispiel: (alle diese Datentypumwandlungen werden von Compiler erlaubt)

```
byte b = 1;  
short s;  
int i;  
long l;  
float f;  
double d;  
s = b;           // short   byte  
i = b [i = s];   // int     byte, short  
l = b [l = s; l = i]; // long   byte, short, int  
f = b [f = s; f = i; f = l]; // float  byte, short, int, long  
d = b [d = s; d = i; d = l; d = f]; // double  byte, short, int, long, float  
  
Abbildung 23
```

Vorsicht! Beachten Sie, dass der Divisions-Operator für zwei int-Werte nur den ganzzahligen Wert ohne Rest zurückgibt.



### Kapitel 3.7.2.10: Arithmetische Operatoren

Mit arithmetischen Operatoren können mathematische Berechnungen durchgeführt werden. Sie erwarten entweder Ganzzahl- bzw. Fließkommazahl-Variablen oder feste Werte als Parameter und liefern ein numerisches Ergebnis zurück.

Aus mehreren Variablen Konstanten kann man mit entsprechenden Operatoren **Ausdrücke** bilden. Zum Beispiel ergibt „**i + 10 + 123**“ die Summe der drei Werte. Ausdrücke können Variablen zugewiesen werden. Die Syntax in Java ist:

**<Variable> = <Ausdruck>;**

Der Ausdruck auf der rechten Seite wird berechnet und dann in die Variable auf der linken Seite gespeichert. Beispiel:

```
int i;  
i = 5;  
i = 5 * 6;
```

Beispiel. Auswertung von **Integerausdrücken**:

Ausdruck	Ergebnis
32 / 5 * 5	30
32 * 5 / 5	32
21 % 6	3 (21 - 3 * 6 )
28 % 7	0
21 / 6	3
23 / 6	3

Die Abkürzung „%“ wird mit „Modulo“ bezeichnet und liefert den Rest einer Division zweier ganzer Zahlen, z.B. 21 geteilt durch 6 ist „3\*6 und der Rest ist 3“, also liefert der Modulo von 21 und 6 die Zahl 3 (21 % 6=3).

In Java gibt es zwei spezielle Operatoren „++“ und „--“ die eine Variable um 1 erhöhen oder erniedrigen. Sie können beispielsweise nicht auf Konstanten angewandt werden (++5 ist nicht erlaubt). Mit „**n = 7**“ wird durch „**i = ++n**“, die Variable **i** auf **8** gesetzt, bei „**i = n++**“, auf „7“. In beiden Fällen hat **n** anschließend den Wert **8**. Man kann die Operatoren anwenden, ohne die Variable weiter zu benutzen, d.h. „**++n**“, oder „**n--**“, sind mögliche Anweisungen und gebräuchliche Abkürzungen für „**n = n + 1**“, bzw. „**n+= 1**“, oder „**n = n - 1**“.

#### Weitere gebräuchliche Abkürzungen:

x+=y	x=x+y
x*=summe	x=x*summe
bestellungen-=summe	bestellungen = bestellungen-summe
n+=1	n=n+1

Beispielprogramm. Das folgende Programm berechnet das Volumen eines Quaders aus den gegebenen Maßen:

```
public class Volumen {  
    public static void main (String [ ] args) {  
        int länge = 8;  
        int breite = 5;  
        int höhe = 3;  
        int volumen = länge * breite * höhe;  
        System.out.println( "Volumen: " + volumen );  
    }  
}
```

### Kapitel 3.7.3: Verkettung von zwei Strings (Z.B. zwei Wörtern)

Beim Datentyp *String* hat der Operator + eine ganz andere Bedeutung. Es verkettet zwei *Strings*. Beispiel: "Hal" + "lo" wird ausgewertet zu "Hallo"

**Aufgabe 11:** Geben Sie die Variablendefinitionen in der Abbildung 20 in Worten wieder. Z.B.: „In der Zeile 2 wird eine Variable mit dem Namen „wahr“ vom Typ boolean definiert und mit dem Wert „true“ initialisiert“.

**Aufgabe 12:** Erweitern Sie das Programm in der Abbildung 18 so, dass alle Variablen aus der Abbildung 20 auf die Konsole ausgegeben werden. Z.B.:

System.out.println("Die Variable mit dem Namen wahr hat den Inhalt:" + wahr);

**Aufgabe 13:** Erweitern Sie das Programm in der Abbildung 18 so, dass folgende Variablen definiert, initialisiert und anschließend auf die Konsole ausgegeben werden:

- Die Variable mit Namen „summe“ vom Typ „double“ mit dem Anfangswert 12.9.
- Die Variable mit Namen „anzahl\_moehren“ vom Typ „int“ mit dem Anfangswert 2329.
- Die Variable mit Namen „einlass“ vom Typ „boolean“ mit dem Anfangswert false.
- Die Variable mit Namen „muster“ vom Typ „char“ mit dem Anfangswert „x“.
- Die Variable mit Namen „anzahl\_zellen“ vom Typ „long“ mit dem Anfangswert 2397812.
- Die Variable mit Namen „das\_jahr“ vom Typ „short“ mit dem Anfangswert 2022.

**Aufgabe 14:** Welche Variablennamen sind zulässig?

beta	ws01/02	Gamma
___test___	dritte_loesung	ws2001_okt_08
ss2001-07-08	3eck	monDay

**Aufgabe 15:** Ergänze Bedeutung und gib die Ergebnisse an.

Operator	Name	Bedeutung	Beispiel	Ergebnis
+	Addition	a + b ergibt die Summe von a und b	a=10; b= 2; c=a + b;	c =
-	Subtraktion	a - b ergibt _____ von a und b	a=10; b= 2; c=a - b	c =
*	Multiplikation	a * b ergibt _____ von a und b	a=10; b= 2; c=a * b	c =
/	Division	a / b ergibt _____ von a und b	a=10; b= 2; c=a / b	c =
%	Modulo	a % b ergibt _____ von a und b	a=10; b= 3; c=a % b	c =
++, --	Präinkrement, Prädekrement	++a erhöht oder verringert (- -a) die Variable a um 1 ____ der weiteren Verwendung	a=10; b= 2; c=++a + b;	a = c=
	Postinkrement, Postdekrement	a++ erhöht oder verringert (a- -) die Variable a um 1 ____ der Verwendung	a=10; b= 2; c=a++ + b;	a = c=

	nt			
+ =,	Zuweisungs	a+ =b weist der Variablen a den Wert	a = 10;	a =
- =;	–operatoren	_____ zu (Kurzschreibweise für	a + = 5;	
* =,		a=_____)		
/ =				

**Aufgabe 16:** Erklären Sie, was beim Kopieren von Daten einer Variablen von einem bestimmten Datentyp in eine andere Variablen von einem ganz anderen Datentyp beachtet werden muss.

**Aufgabe 17:** Schreiben Sie ein Programm, dass primitive Datentypen umwandelt. Folgende Umwandlungen soll das Programm durchführen und die Inhalte der Variablen auf die Konsole ausgeben (Siehe dazu Abbildung 23):

short    byte	long    byte	float    byte, short, int,	double    int
int    byte, short	long    short	long	double    long
int    short	long    int	float    int	

**Aufgabe 18:** Wie kann die folgende arithmetische Operation  $zaehler = zaehler + 1$  vereinfacht werden? Nenne mindestens drei verschiedene Varianten!

**Aufgabe 19:** Wenn arithmetische Operatoren mit Variablen verschiedener Datentypen durchgeführt werden, wird der Datentyp für das Ergebnis nach bestimmten Regeln festgelegt. Gebe jeweils an, welcher Datentyp aus den folgenden Operationen resultiert:

a) int + int	b) long + int	c) float + double	d) float + byte
--------------	---------------	-------------------	-----------------

**Aufgabe 20:** Auf welchen Datentypen kann der Modulo-Operator ausgeführt werden?

**Aufgabe 21:** Welche Werte haben  $i$ ,  $j$  und  $k$  nach der Durchführung der folgenden Operationen? Ermitteln Sie das Ergebnis ohne ein Programm dafür zu schreiben.

```
int i, j, k;
i=5;
j=7;
k=3;
k *= - - i * j ++;
```

**Aufgabe 22:** Wie werden die Ausdrücke ausgewertet? Von welchen Datentypen sind die Ergebnisse? Klammern haben die gleiche Bedeutung wie in der Mathematik.

	<i>Ergebnis</i>	<i>Datentyp des Ergebnisses</i>
23.4 + 7	30.4	double
(10 / 3) + 0.5		
'a' + 'b'		
"text" + "Text"		
"Pro" + "gramm"		
33 - 20		
6.6 / 3.3		
(1 / 3) * 1234567891234		
'Q' + „quit“		

### Und ein paar etwas komplizierteren Aufgaben.

**Aufgabe 23:** Beschriften Sie das Programm mit den folgenden Begriffen: Kommentare, Hauptprogramm, Anweisungen, Deklarationen

```
public class Kreisumfang
{
    public static void main(String[] args) {
        //Ausgabe der Ueberschrift

        System.out.println("\n KREISUMFANG \n");
        //Berechnung des Umfangs
        double r=5;
        double u;
        System.out.println(" r | Kreisumfang ");
        u = r * 2.0 * 3.14159;
        System.out.print(" ");
        System.out.print(r);
        System.out.print(" | ");
        System.out.println(u);  }}
```

Abbildung 24

**Aufgabe 24:** Geben Sie an, welche Ausgaben die folgenden Programme (Abbildung 25 bis Abbildung 27) am Bildschirm erzeugen. Geben Sie anschließend den Javaprogrammcode in den JavaEditor ein und überprüfen Sie Ihre Ergebnisse. Kommentieren Sie Ihre Programme, d.H. Was sind hier die Variablen und von welchen Typ sind sie? Unter welchen Namen müssen die Programme gespeichert werden?

```
public class Mathe1 {
    public static void main(String[] args) {
        int a=1+1;
        int b=a*3;
        int c=b/4;
        int d=b-a;
        int e=-d;
        int x=42;
        float y=42.3f;
        System.out.println("a=" +a);
        System.out.println("b=" +b);
        System.out.println("c=" +c);
        System.out.println("d=" +d);
        System.out.println("e=" +e);
        System.out.println("x=" +x%10);
        System.out.println("y=" +y%10);
    }
}
```

Abbildung 25 Programm 1.

```
public class Mathe2 {
    public static void main(String[] args) {
        int a=1;
        int b=2;
        int c=3;
        a +=5;
        b *=4;
        c %=6;
        System.out.println("a = " +a);
        System.out.println("b = " +b);
        System.out.println("c = " +c);} }
```

Abbildung 26 Programm 2.

```

public class Mathe3 {
    public static void main(String[] args) {
        int a=1;
        int b=2;
        int c= ++b;
        int d= a++;
        c ++;
        System.out.println("a=" +a);
        System.out.println("b=" +b);
        System.out.println("c=" +c);
        System.out.println("d=" +d);
    }
}

```

Abbildung 27 Programm 3.

### Kapitel 3.7.3.1: Inhalte von Variablen ändern

Der Wert einer Variablen kann beliebig oft geändert werden. In den meisten Programmen werden Variablen nicht angelegt, um im Ablauf des Programms den selben Wert zu behalten. Wie der Name schon sagt, ist der Speicherplatz einer Variablen „variabel“. Um den Wert zu ändern, muss mit einem Gleichzeichen ein neuer Wert zugewiesen werden. Beachten Sie, dass immer von rechts nach links zugewiesen wird. Beispiel:

```

//Datei Zuweisung.java
public class Zuweisung{
    public static void main(String[] args){
        byte ersteZahl = 25;
        byte zweiteZahl = 2;
        zweiteZahl = ersteZahl;
        ersteZahl = 18;
        15=zweiteZahl; //Fehler
    }
}

```

Abbildung 28

Mit einem Gleichzeichen wird hier der Variablen „ersteZahl“ die 18 neu zugewiesen. Alternativ besteht die Möglichkeit, den Wert einer bereits vorhandenen Variablen zuzuweisen.

Durch den Befehl „zweiteZahl = ersteZahl;“ wird in der Variablen „zweiteZahl“ der Wert von „ersteZahl“, also 18, gespeichert.

Die Anweisung „15=zweiteZahl;“ geht nicht!!! Dies führt sofort zu einer Fehlermeldung.

### Kapitel 3.7.3.2: Konstanten

Wie Sie gelernt haben, lässt sich der Wert einer Variablen durch Zuweisungen verändern. Bei manchen Variablen ist es jedoch notwendig, die Zuweisung eines Wertes nach der Initialisierung zu verbieten:

```

//Datei Konstanten.java
public class Konstanten {
    public static void main(String[] args) {
        final double PI = 3.141592654;
        final byte MWST = 19;
        PI = 23; //Verboten
    }
}

```

Abbildung 29

Um zu verhindern, dass der Wert einer Variablen verändert wird, verwendet man das Wort „**final**“. Wird eine Variable mit „**final**“ deklariert, spricht man nicht mehr von einer Variablen, sondern von einer Konstanten. Im Gegensatz zu Variablen sollte eine Konstante immer in der Zeile in der sie deklariert wird auch initialisiert werden. Der Name einer Konstanten wird ausschließlich in großen Buchstaben geschrieben.

### Kapitel 3.7.3.3: Variablen und der Ausgabebefehl

Variablen können auf dem Bildschirm dargestellt werden. Dafür muss der Name einer Variablen in die Klammer des Befehls `System.out.println()`; geschrieben werden:

```
//Datei Ausgabe.java
public class Ausgabe {
    public static void main(String[] args) {
        final double PI = 3.141592654;
        System.out.println("PI hat den Wert: " +PI);
        System.out.println(PI+ " ist der Wert von PI");
        System.out.println("Der Wert " +PI+ " ist PI");
    }
}
```

Abbildung 30

Mit einem Pluszeichen (+) kann die Variable mit beliebigem Text kombiniert werden. Steht die Variable zwischen zwei Text-Elementen, muss das Plus (+) auf beiden Seiten des Variablennamens stehen. Wichtig ist, dass man nie vergisst, den Text in Anführungszeichen zu stellen. Wenn man die Anführungszeichen vergisst, denkt Java, dass es sich um eine Variable handelt.

### Kapitel 3.8: Einfache Benutzereingaben vom Programm einlesen lassen .

Zu Beginn eines Quelltextes mit Eingabe muss mit Hilfe der import-Anweisung die Klasse `java.util.Scanner` (eine Klasse zum Scannen der Tastatur) bekannt gemacht werden. Dies geschieht mit der Zeile

```
import java.util.Scanner;
```

bzw.

```
import java.util.*;
```

am Programmanfang. Zusätzlich muss ein Scanner erzeugt werden. Dies geschieht mit folgenden Zeilen vor der ersten Eingabe:

```
Scanner Eingabe = new Scanner (System.in);
```

#### Beispiel 1: Einlesen einer ganzen und einer Fließkommazahl:

```
int a = Eingabe.nextInt ();
double b = Eingabe.nextDouble ();
```

#### Beispiel 2: Einlese einer String-Wertes:

```
System.out.print ("Bitte Name eingeben: \n ");
String Name = Eingabe.next();
```

#### Beispiel 2: Einlese eines char-Wertes:

```
// Zeichen lesen und in char umwandeln
String line = Eingabe.next();
char auswahl = line.charAt(0);
```

**Aufgabe 25:** Alter ausrechnen. Schreibe ein Programm, das den Namen, das aktuelle Jahr und das Geburtsjahr des Benutzers einliest und das Alter des Benutzers ausgibt. Dazu sollte es nachgefragt werden, ob der Benutzer in diesem Jahr schon Geburtstag hatte. Der Dialog, der bei der Ausführung des Programms entsteht, könnte zum Beispiel so aussehen

```
Hallo, wie heißt du?
Mia
Welches Jahr haben wir, Mia?
Jahr: 2010
In welchem Jahr bist du geboren?
Jahr: 1993
Hattest du dieses Jahr schon Geburtstag?
Dann bist du 16 Jahre alt.
```

**Aufgabe 26:** Spritkostenberechnung. Schreibe ein Programm zur Spritkostenberechnung!

Das Programm liest den aktuellen Spritpreis, den durchschnittlichen Spritverbrauch des Autos pro 100 km und die Anzahl der zu fahrenden Kilometer ein. Das Programm gibt aus wie teuer die Fahrt wird. Beispiel für die Eingaben/Ausgaben während der Ausführung des Programms

```
****Benzinkostenrechner****
aktueller Spritpreis in Cent: 137
Verbrauch des Autos pro 100 km in Liter: 8
Zu fahrende Strecke: 100
*****
Die Fahrt kostet 10.92 Euro.
```

Was würde es kosten, mit dem Auto von Neuss nach Madrid, Istanbul, Peking und Köln zu fahren? Finde dazu im Internet die aktuellen Spritpreise, wie viel Liter Benzin dein Auto bzw. das Auto von deinen Eltern pro 100 km verbraucht und die entsprechenden Entfernungen heraus.

**Aufgabe 27:** Erstelle einen einfachen Taschenrechner mit deren Hilfe du Summe, Differenz, Produkt und Quotient von zwei Ganz- und Fließkoma-Zahlen berechnen kannst. Alle Rechnungen werden nacheinander durchgeführt. Beide Zahlen sollen eingelesen werden und die Ergebnisse der Berechnungen sollen auf der Konsole zu sehen sein.

## Kapitel 3.9: Kontrollstrukturen

Kontrollstrukturen sind Programmelemente, die die Ausführung von Anweisungen steuern, insbesondere durch Verzweigung und Wiederholung.

### Kapitel 3.10: Verzweigungen

Die Syntax für eine Verzweigung kann auf verschiedene Weise formuliert werden. Zunächst schauen wir uns die am häufigsten verwendete Verzweigung „if“ an. In dem meisten Fällen geht es darum, ob eine Bedingung erfüllt ist oder nicht. Wir haben demnach eine Straßenkreuzung mit zwei Wegen vor uns und entscheiden, welchem Weg wir folgen.

Die zweite Verzweigungsmöglichkeit **switch–case** kann als mehrspurige Autobahn angesehen werden und wir entscheiden, auf welcher der vielen Spuren wir nun weiterfahren möchten.

#### Kapitel 3.10.1: Vergleichsoperationen

Für die Struktur der Fallunterscheidung werden Vergleichsoperatoren benötigt. Um zwei Werte miteinander zu vergleichen gibt es folgende Vergleichsoperationen.

<	kleiner
<=	kleiner oder gleich (kleinergleich)
>	größer
>=	größer oder gleich (größergleich)
==	gleich
!=	ungleich

Diese Operationen können natürlich nur auf Zahlen angewendet werden. Sollen Buchstaben verglichen werden, dann kann nur der Operator „==“ verwendet werden. Sollen Wörter (Also String-Variablen) verglichen werden, dann verwendet man die String-Methode „equals(...)“.

Beispiel:

```
String text1 = „Tach“;
String text2= „Hallo“;
if (text1.equals(text2))      //Das Ergebnis des Vergleichs „text1.equals(text2)“ liefert den Wert „false“
{                             //Also nicht gleich.
    //Anweisungen
}
```

Beispiel:

```
char buchstabe1 = „T“;
char buchstabe2= „H“;
if (buchstabe1==buchstabe2)  //Das Ergebnis des Vergleichs „buchstabe1==buchstabe2“ liefert  }
{                             //den Wert „false“Also nicht gleich
    //Anweisungen
}
```

### Kapitel 3.10.2: Logische Operatoren

Um mehrere Bedingungen miteinander zu verknüpfen benötigt man logische Operatoren (Siehe Beispiel in der Abbildung 38).

Operator	Bezeichnung	Beispiel	Beschreibung
!	Nicht	!b	Es wird a negiert. Ist a wahr, wird false zurückgegeben, andernfalls true.
&&	Und	a&& b	Es wird a und b ausgewertet. Die Auswertung wird abgebrochen wenn a bereits falsch ist. Dann wird sofort false zurückgegeben. Sind beide wahr, wird true zurückgegeben, andernfalls false.
	Oder	a  b	Es wird a und b ausgewertet. Die Auswertung wird abgebrochen wenn a bereits wahr ist. Dann wird sofort true zurückgegeben. Ist mindestens einer der beiden Ausdrücke wahr, wird true zurückgegeben, andernfalls false.



### Kapitel 3.10.3: Klassische Verzweigung mit if

Soll nur eine Anweisung ausgeführt werden, dann lautet die Syntax: `if (Bedingung) <Anweisung>;`

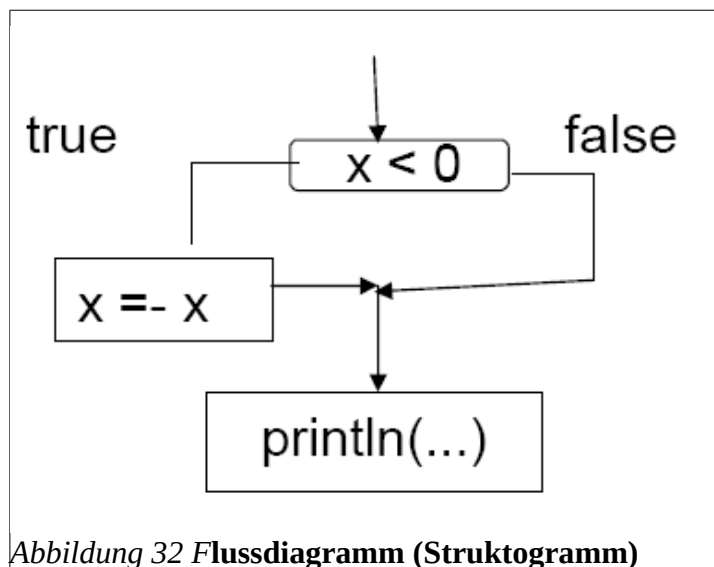
Beispiel 1:

`if (a < b) a = b;` // Wenn  $a \geq b$  ist, bleibt  $a$  unverändert und wenn  $a < b$  ist, dann ist  $a = b$

Beispiel 2:

`if (x < 0) x = -x;` // Wenn  $x < 0$  ist, wird Anweisung  $x = -x$  ausgeführt  
`System.out.println("x: " + x);` // Wenn  $x \geq 0$  ist, wird die Anweisung `System.out.println("x: " + x);`

Abbildung 31



Es können aber mehrere Anweisungen, ein ganzer Block von Anweisungen, ausgeführt werden. Dazu verwenden wir wieder die geschweiften Klammern:

```
if (Bedingung)
{<Anweisungen>; }
```

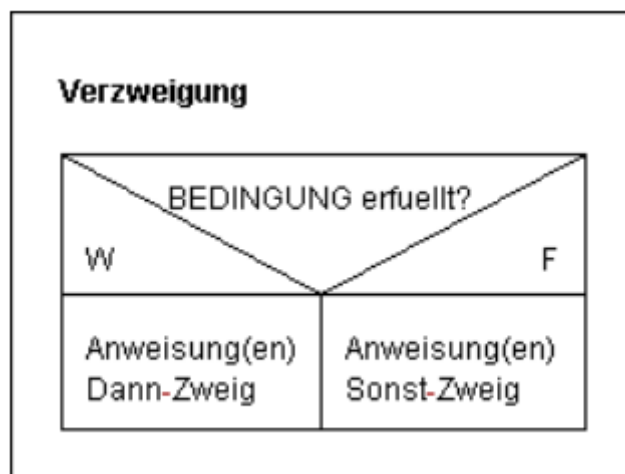
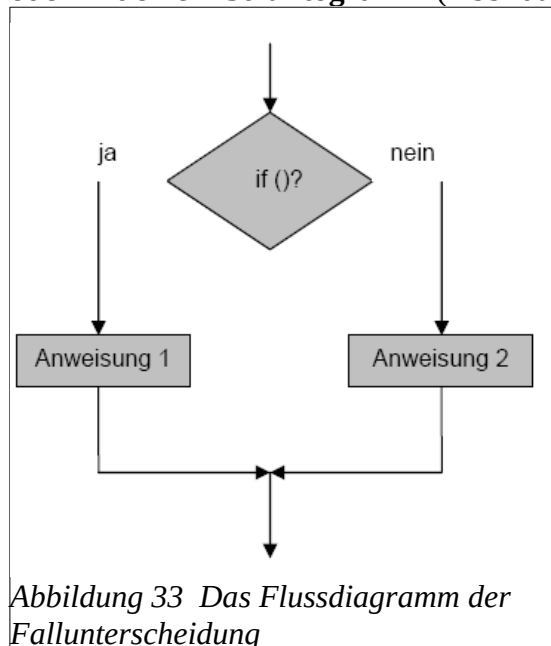
Eine Erweiterung dazu ist die **if-else** Verzweigung. Wenn eine Bedingung erfüllt ist, führe eine **Anweisung 1** aus. Wenn diese Bedingung nicht erfüllt ist, dann führe **Anweisung 2** aus. In diesem Fall wird eine der beiden Anweisungen ausgeführt, weil die Bedingung ist entweder erfüllt oder nicht.

```
if (Bedingung)
    <Anweisung 1>;
else
    <Anweisung 2>;
```

Beispiel 1: <code>if (x &gt; 0)</code> <code>System.out.println("x: " + x);</code> <code>else</code>	Beispiel 2: <code>if (zahl &lt; 0) {</code> <code>System.out.print("Zahl ist negativ");</code> <code>}else</code>
---	--

System.out.println("x: " + (-x));	{ System.out.print("Zahl ist nicht negativ");}
Beispiel 3: (mit der Anweisungsblock): if (b > 2) { c = 4; System.out.println(c); } else { c = 0; System.out.println("Fehler"); }	

Das Verhalten und die Wirkung der Fallunterscheidung lässt sich durch ein **Flussdiagramm** (Abbildung 33) **oder mit einem Struktogramm** (Abbildung 34) beschreiben:



**if**-Anweisungen können verschachtelt werden.

```

if(<Bedingung>)
{
    if(<Bedingung_2>)
    {
        <Anweisungen>;
    }
} else
{
    <Anweisungen>;
}

```

Abbildung 36

Beispiel:

```

if (x < 0) { //Wenn x < 0 ist, wird -x in betrag gespeichert
    betrag=-x;
}
else{ //Sonst..
    if (x > 0) { //..ist entweder x > 0,
        betrag=x; } //dann wird x in betrag gespeichert
    else{ //..oder x = 0,
        betrag=0; //dann wird 0 in betrag gespeichert
    }
}

```

Abbildung 35

**!!! Unterscheide:**

<pre> if (x &gt; 0)     if (x &lt; 10)         x = x + 1; else     x = -x; </pre>	äquivalent zum folgendem Flussdiagramm:
---	---

	<pre> graph TD     Entry(( )) --&gt; Cond1{x &gt; 0}     Cond1 -- true --&gt; Cond2{x &lt; 10}     Cond1 -- false --&gt; Exit(( ))     Cond2 -- true --&gt; Act1{x =+ 1}     Cond2 -- false --&gt; Act2{x =- x}     Act1 --&gt; Exit     Act2 --&gt; Exit </pre>
<pre> if (x &gt; 0) {     if (x &lt; 10)     x = x + 1; } else    x = -x; </pre>	<p>denn: <b>else</b> wird auf das letzte "ungesättigte" <b>if</b> bezogen</p> <p>äquivalent zum folgendem Flussdiagramm:</p> <pre> graph TD     Entry(( )) --&gt; Cond1{x &gt; 0}     Cond1 -- true --&gt; Cond2{x &lt; 10}     Cond1 -- false --&gt; Act2{x =- x}     Cond2 -- true --&gt; Act1{x =+ 1}     Cond2 -- false --&gt; Act2     Act1 --&gt; Exit(( ))     Act2 --&gt; Exit </pre>

Es können auch eine Serie von Bedingungen geprüft werden. Bei einer Mehrfachentscheidung werden mehrere „ifs“ verwendet, bei denen die Anweisung, die zu jedem else gehört, wieder ein if ist.

```

if (erster Bedingung) <Anweisung 1>;
else if (zweiter Bedingung) <Anweisung 2>;
...
else if (letzter Bedingung) <letzte Anweisung>;
else <Fehlansweisung>;

```

Abbildung 37

Beispiel:

```

if (monat >= 3 && monat <= 5) System.out.println("Frühling");
else if (monat >= 6 && monat <= 8) System.out.println("Sommer");
else if (monat >= 9 && monat <= 11) System.out.println("Herbst");
else if (monat == 12 || monat == 1 || monat == 2)
System.out.println("Winter");
else System.out.println("Ungültige Eingabe");

```

Abbildung 38 Hierbei wird der logische Und-Operator „&&“ und der Oder-Operator „||“ verwendet.

**Aufgabe 28:** In der folgenden Tabelle sollen die Bedingungen der Kontrollstruktur „if“ ausgewertet werden:

	<i>Ergebnis der Prüfung der Bedingung</i>
Int a=10; int b=5 if(a<b)	false
Int a=10; int b=5 if(a>b)	
Int index=-5; int c=5; if(a==b)	
Int index=-5; int c=5; if(index<=c)	
double summe=-5.23; double c=-5.24; if(summe>=c)	
String index="stop"; String punkt = "weiter"; if(index.equals(punkt))	
String index=new String("stop"); String punkt = new String("stop"); if(index.equals(punkt))	
Int index=-5; int c=5; if(index<=0 && c>2)	
Int index=-5; int c=5; if(index<=0    c>2)	
Int index=15; int c=5; if(index<=0 && c>2)	
Int index=15; int c=5; if(index>0    c<=2)	
Int index=15; int c=5; double z=3.1415 if((index>=-9 && c<=6)    z==3.1416)	

**Aufgabe 29:** Folgendes Struktogramm ist Ihnen gegeben (Abbildung 39). Erstellen Sie aus dem Struktogramm das entsprechende Programm. Mit „Attribut“ ist hier eine Variable gemeint.

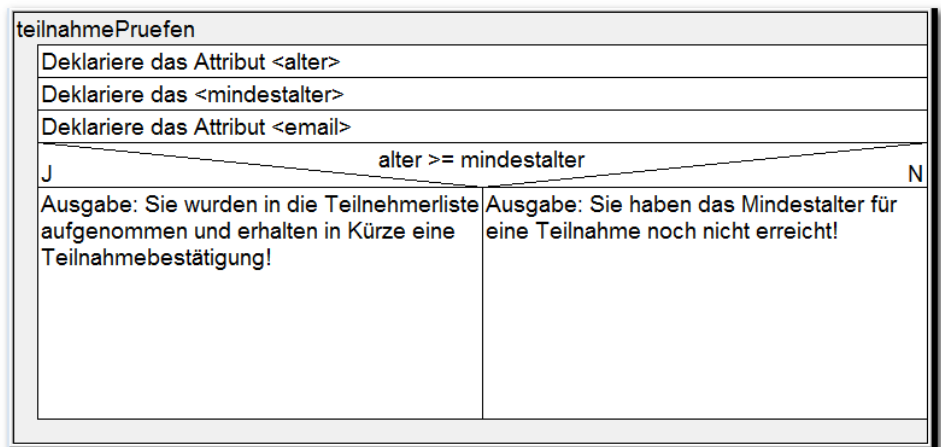
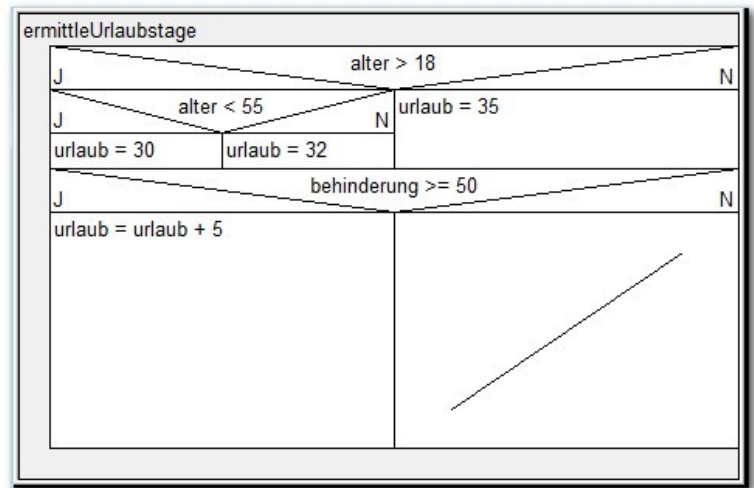


Abbildung 39

**Aufgabe 30:** Die Betriebsvereinbarung der Beschäftigten enthält folgenden Auszug: "Den Beschäftigten stehen im Jahr 30 Tage Urlaub zu. Minderjährige erhalten 35 Tage Urlaub, Beschäftigte, die älter als 55 Jahre sind, 32 Tage. Zusätzlich zu ihrem Urlaubsanspruch erhalten Beschäftigte mit einer Behinderung ab 50 % weitere fünf Urlaubstage. ". Das Struktogramm in der Abbildung 40 zeigt die Arbeitsweise des Programms. Setzen Sie das Struktogramm in Java um. Implementieren Sie auch eine entsprechende Ausgabe.



**Aufgabe 31:** Das Programm soll Krankenhaustagegeld ermitteln. Entwickeln Sie *Abbildung 40* den entsprechenden Quellcode und das dazugehörige Struktogramm und das Flussdiagramm:

Die Idunal Versicherung versichert auch Zusatzleistungen, dazu gehört u.a. das Krankenhaustagegeld. Patienten erhalten ein Krankenhaustagegeld in Höhe von 11 € pro Tag. Patienten deren Aufenthaltsdauer eine Woche (7 Tage) nicht überschreitet erhalten ein Krankenhaustagegeld von 10 €, Patienten deren Aufenthalt den Zeitraum von 21 Tagen überschreitet, erhalten 12 €. Zusätzlich zu ihrem Krankenhaustagegeld erhalten privat versicherte Patienten einen zusätzlichen Betrag von 2 € pro Tag.

**Aufgabe 32:** Das Struktogramm in der Abbildung 41 zeigt, wie je nach Höhe des Umsatzes dem Kunden ein bestimmter Rabatt gewährt wird. Implementieren Sie das entsprechende Programm zu diesem Struktogramm und formulieren Sie in Worten die Arbeitsweise des Programms.

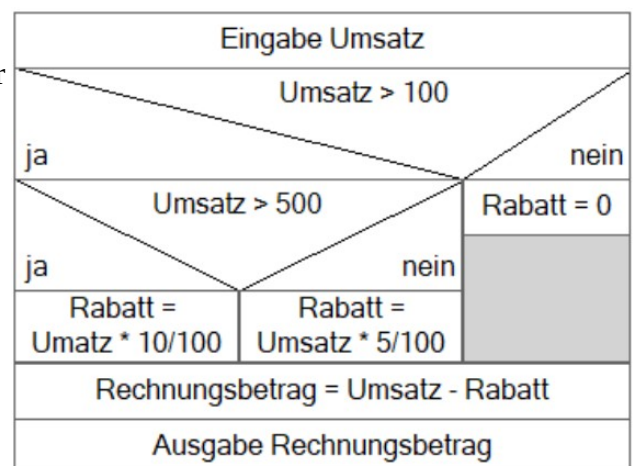


Abbildung 41

## Kapitel 3.11: Verzweigung mit switch

Wenn bei einer Mehrwegentscheidung alle Vergleiche auf verschiedene Werte desselben Ausdrucks geprüft werden, ist die **switch-Anweisung** gut geeignet.

```
switch (<Ausdruck>) {  
    case <Konstante 1>:  
        <Anweisung 1>;  
        break;  
    case <Konstante 2>:  
        <Anweisung 2>;  
        break;  
    ...  
    case <Konstante n>:  
        <Anweisung n>;  
        break;  
    default:  
        <Fehlanweisung>;  
}
```

Abbildung 42 Die allgemeine Syntax einer switch-Anweisung

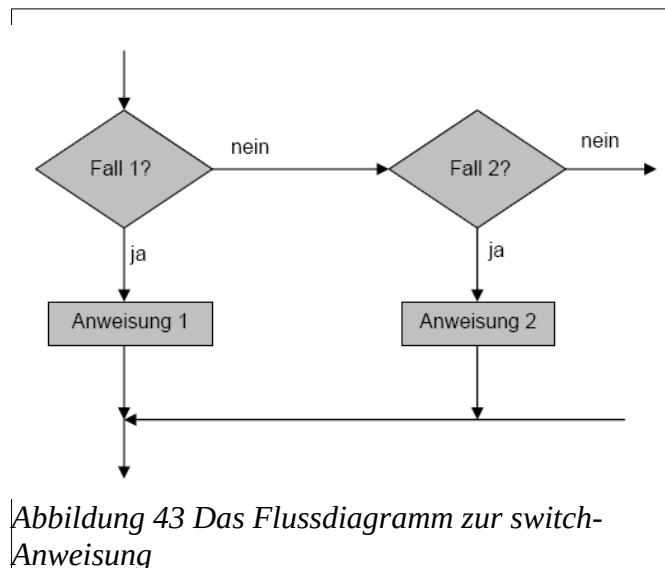


Abbildung 43 Das Flussdiagramm zur switch-Anweisung

Die **switch**-Anweisung wird nicht durch eine Bedingung von Typ **boolean** gesteuert, sondern von einem Ausdruck, der Werte vom Typ **byte**, **short**, **char** oder **int** haben muss.

**case** ist ein Sprungziel. Stimmt der Wert von **Ausdruck** in der **switch**-Anweisung mit der Konstanten eines **case**-Ziels überein, werden alle Anweisungen nach diesem **case**-Ziel ausgeführt. Stimmt der Wert von **Ausdruck** mit keiner Konstanten eines **case**-Ziels überein, werden die Anweisungen nach dem **default**-Ziel ausgeführt. Der **default**-Zweig ist optional und kann entfallen.

Beispiel 1:

Programmcode	Bedeutung
<pre>switch (x){     case 0:         System.out.println („0“);         break;     case 1:     case 2:         System.out.println („1 oder 2“);         break;     default : {         System.out.println („hier landen alle anderen...“);     } }</pre>	<p>Falls x gleich 0 ist, dann wird case ausgeführt und mit dem <i>break</i> signalisiert, dass die switch-Verzweigung beendet wird.</p> <p>Die Fälle case 1 und case 2 führen für x=1, 2 zu der selben Ausgabe, denn ein <i>break</i> beendet ein case.</p> <p>Für alle anderen Werte für x, die nicht durch ein case abgedeckt sind, tritt der default-Block in Kraft.</p>

Beispiel 2:

```
switch (i) {  
    case 0: System.out.println("Null"); break;  
    case 1: System.out.println("Eins"); break;  
    case 2: System.out.println("Zwei"); break;  
    case 3: System.out.println("Drei"); break;  
    default: System.out.println("größer Drei"); }  
}
```

Abbildung 44

Die **switch**-Anweisung kann auch verwendet werden, um in Abhängigkeit einer Eingabe von der Tastatur den Programmablauf zu verzweigen.

Beispiel:

```
char zeichen;
zeichen von Tastatur einlesen
switch (zeichen) {
    case 'A': anweisungen; break;
    case 'B': anweisungen; break;
    case 'C': anweisungen; break;
    case 'D': anweisungen; break;
    default: System.out.println("Falsche Eingabe"); }
```

**Aufgabe 33:** Für ein Zeugnis (Siehe Abbildung 45) wird der Ziffernwert einer Note eingelesen (simulieren Sie die Eingabe durch Initialisierung der entsprechenden Variablen). Anhand des Wertes soll ein Text gedruckt werden, der den Notenwert wiedergibt (Z. B. Note 1 = sehr gut). Entwickeln Sie dazu ein Java-Programm unter Verwendung einer switch-Anweisung. Vergessen Sie nicht die Ausgabe.

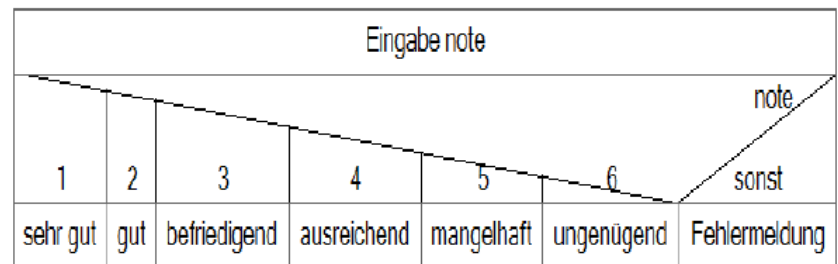


Abbildung 45 Das Struktogramm der Switch-Anweisung

**Aufgabe 34:** Implementieren Sie zum Struktogramm in der Abbildung 46 das entsprechende Java-Programm. Mit einer Ausgabe! Formulieren Sie mit eigenen Worten die Arbeitsweise des Programms.

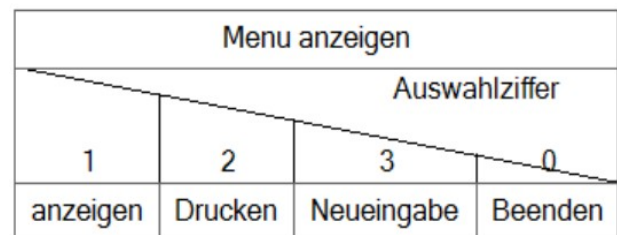


Abbildung 46

**Aufgabe 35:** Erstelle ein Programm, dass für eine Zahl, die über die Tastatur eingelesen wird, ausgibt, ob die Zahl negativ oder positiv ist! Sorge durch Einrückungen und Kommentaren für eine gute Lesbarkeit deines Programms.

**Aufgabe 36:** Der Kunde, der sich gerade am Bankautomat mit seiner PIN legitimiert hat, hat ein Guthaben von 136.34 €. Das Konto kann nicht überzogen werden. Erstelle ein Programm, das einliest, wie viel Geld dieser Kunde abheben möchte, und nur dann Geld ausgibt, wenn dieser Betrag das Guthaben nicht übersteigt.

Beispiele:

*Ihr Guthaben beträgt 136.34 Euro.  
Wie viel Geld wollen Sie abheben? 200  
Keine Auszahlung! Dieser Betrag übersteigt Ihr Guthaben.*

*Ihr Guthaben beträgt 136.34 Euro.  
Wie viel Geld wollen Sie abheben? 100*

**Aufgabe 37:** Erstelle ein Programm, das drei int-Werte einliest und den größeren und den kleineren ausgibt.

**Aufgabe 38:** Erstelle ein Programm, das eine Zahl einliest und den zugehörigen Kehrwert ausgibt. Zusätzlich soll überprüft werden, ob die eingegebene Zahl 0 war. In diesem Fall soll eine Meldung ausgegeben werden, dass kein Kehrwert gebildet werden kann.

**Aufgabe 39:** Wie kann man das folgende Listing vereinfachen?

```
if(zahl == 0) System.out.println(„Kein Kehrwert“);
if(zahl != 0) System.out.println(„1/zahl“);
```

**Aufgabe 40:** Erstellen Sie ein Programm, dass von Tastatur eine beliebige Zeichenfolge einliest und prüft, ob es sich dabei um eine Zahl zwischen 1 und 5 handelt. Liegt die Zahl in diesem Bereich, ist sie auszugeben, ansonsten soll eine entsprechende Meldung erscheinen. Die Aufgabe ist mithilfe der *switch-case-Anweisung* zu realisieren.

**Aufgabe 41:** Erstellen Sie ein Programm, dass von der Tastatur eine Zahl einliest und danach prüft, ob es sie größer, kleiner oder gleich Null ist. Außerdem soll ausgegeben werden, ob es sich um eine gerade oder eine ungerade Zahl handelt.

**Aufgabe 42:** Welche Ausgabe erzeugt das folgende Programm? Ist sie korrekt? Wenn nicht, korrigieren Sie das Programm!

```
int a = 5, b=7 c = 4;
if (a < b)
    if (a < c)
        System.out.println („a ist die kleinste Zahl“);
    else
        System.out.println („a ist nicht kleiner als b“);
```

**Aufgabe 43:** Erstellen Sie ein Java-Programm zur Berechnung aller möglichen Nullstellen einer beliebigen quadratischen Gleichung  $ax^2 + bx + c = k$ . Die allgemeine Normalform einer quadratischen Gleichung lautet  $x^2 + p x + q = 0$ . Nullstellen lassen sich nun mit der so genannten pq-Formel berechnen. Da eine quadratische Gleichung zwei reelle, eine Doppelte oder zwei komplexe Nullstellen haben kann, berechnet man zuerst die Diskriminante:

$$D = p^2 / 4 - q.$$

Ist die Diskriminante größer als der Wert 0 ( $D > 0$ ), dann gibt es zwei reelle Nullstellen.

Ist die Diskriminante gleich dem Wert 0 ( $D = 0$ ), dann gibt es eine doppelte Nullstelle.

Ist die Diskriminante kleiner als der Wert 0 ( $D < 0$ ), dann gibt es keine reelle Nullstelle. Die Struktur der Fallunterscheidung können Sie aus der Abbildung 47 entnehmen.

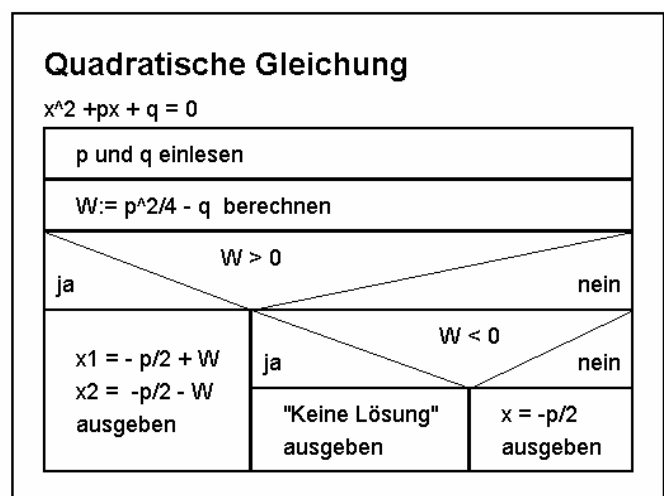


Abbildung 47 Das Struktogramm

**Aufgabe 44:** Erstelle ein Programm für die Funktion, die in der Abbildung 48 beschrieben ist.



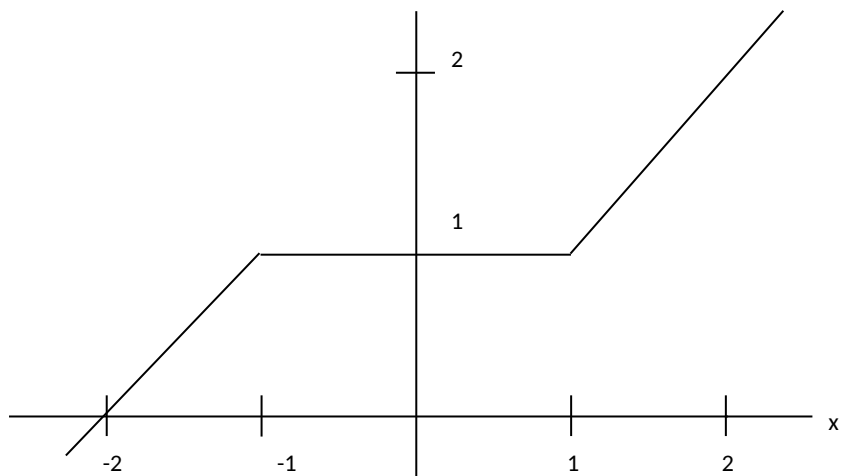


Abbildung 48:

## Kapitel 4: Beispielklausur Nr. 1: Thema: Primitive Datentypen, Variablen, Kontrollstrukturen I.

### Aufgabe 1. Java-Grundlagen

a) Theoretische Fragen (Siehe Aufgaben im Script; Oder Fragen zum Text im Script)

### Aufgabe 2. Variablen

a) Stellen Sie fest, welchen Wert jeweils die **Variablen a, b, c, d und k** am Ende der angegebenen Anweisungsfolge haben, wenn **sie alle zunächst mit dem Wert 2 initialisiert wurden**.

```
a+=2;
b*=5;
c+=++a*b--;
c%=5;
d= d+(b++)*b;
```

b) Welchen Wert bekommen die Variablen **li** und **mi** vor der Zeile 2 und nach der Zeile 2?

```
1) int mi = 14; int li = 12; mi = li; li = mi;
2) li = mi + 2;
```

c) Welche der hier angegebenen Java Code-Schnipsel sind syntaktisch falsch bzw. welche halten Sie für kritisch bei der Programmausführung? Begründen Sie Ihre Antwort kurz!

**A:** char a="f";  
int while=7;  
int a=3k2;

**B:** String name;  
System.out.println(„Ihr Name: „, +name);

**C:** int i=15;  
if(i>10) j=i+5;  
else i=i-5;

d) Wie werden die Ausdrücke ausgewertet? Von welchen Datentypen sind die Ergebnisse? Klammern haben die gleiche Bedeutung wie in der Mathematik.

	<i>Ergebnis</i>	<i>Datentyp des Ergebnisses</i>
-2 + 12.8		
(12.5 + 4.3) * 0.5		
"Klausur" == "klausur"		
"Info" + "kurs"		
"nein" != "Nein"		
'Q' == 'q'		
("Info" + "rmatik") == "Mathematik"		
10 <= (20/2)		
100 > 200		
(1.0 + 2) != 3.0		

Aufgabe 5. **Kontrollstrukturen (Die Nummerierung der Aufgaben ist defekt?!?!?! Bug in libreoffice?!?!)**

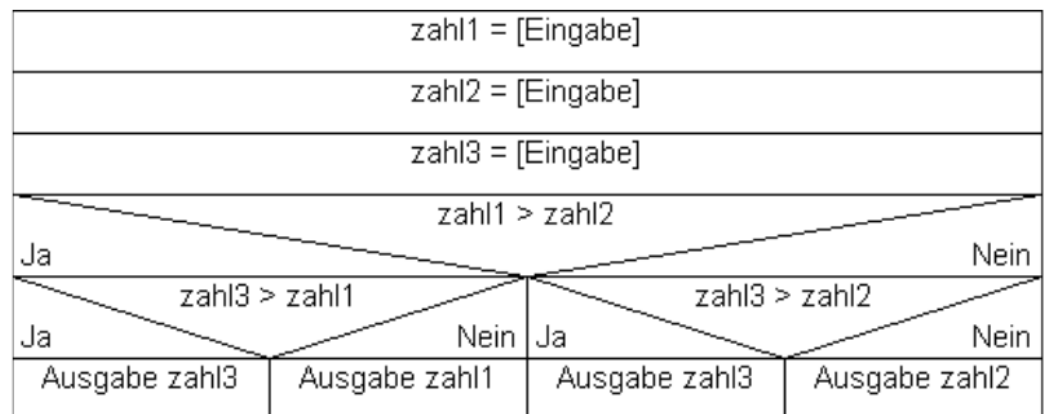
- a) Ersetzen Sie die *if*-Konstruktionen durch eine einzige *if*-Anweisung und geben Sie das dazugehörige Struktogramm (und das dazugehörige Flussdiagramm) an. (Die Variablen x und y sind vom Typ **int**.)

```

if (y > x) {}
else {
    if (y < x) {}
    else
        System.out.println("Aufgabe 4");
}
    
```

Aufgabe 2. **(Die Nummerierung der Aufgaben ist defekt?!?!?! Bug in libreoffice?!?!)** Zeichnen Sie ein Struktogramm für folgende Problemstellung: Es wird eine Zahl über die Tastatur eingegeben. Wenn die Zahl gerade ist, wird sie mit 2 multipliziert, wenn sie ungerade ist, wird zu dieser Zahl der Wert 1 addiert. Anschließend wird das Ergebnis ausgegeben.

Aufgabe 3. Das folgende Struktogramm zeigt den Ablauf eines Programms. Erklären Sie, was das Programm berechnet (herausfindet) und Geben das dazugehörige Javaprogramm an.



Das folgende Programm ist gegeben:

```
01  public class Aufgabe_5 {
02  public static void main(String[] args) 03  {
04  int posx = 10;
05  int posy = 110;
06  if (posx != posy)
07  {
08      posy = posy -100;
09      if ((posx-9) == posy)
10      {
11          posy = 110 - 60;
12          System.out.print( "posy: " + posy);
13      }
14      else {
15          posx = posx * 10;
19          System.out.print( "posx: " + posx);
20      }
21  }
22  else {
23      posy = posx + 250;
24      System.out.print( "posy: " + posy);
25      }
26  }
27  }
```

- a) Kommentieren Sie das oben gegebene Programm (jede Zeile der Programmcode) in Ihrem Heft sinnvoll.
- b) Was wird in den Zeilen 12, 19 und 24 ausgegeben und unter welchen Bedingungen findet eine Ausgabe statt. Begründen Sie!
- c) Verändern Sie den Quelltext des *Programms* so, dass die Variablen **posx** und **posy** von dem Benutzer (**Scanner und entsprechende Kommentare**) eingegeben werden können. Was wird nach der Ausführung das Programm ausgegeben, wenn für die Variable **posx** folgende Werte eingegeben werden: **posx =19;** und **posx =110**. Geben Sie die Zeilennummer an, in der entsprechende Ausgabeanweisung steht.

```
import java.util.Scanner;
Scanner Eingabe = new Scanner (System.in);
int a = Eingabe.nextInt ();
double b = Eingabe.nextDouble ();
String Name = Eingabe.next();
```

Aufgabe 4. Schreiben Sie ein „Happy World“-Programm, welches die folgenden beiden Zeilen auf der Konsole ausgibt:

Happy World

Die Klausur ist geschafft!

## Kapitel 5: Schleifen

Schleifen sind auch wie Verzweigungen Kontrollstrukturen. Sie bewirken eine Abweichung von der üblichen Abarbeitung der Anweisungen. Solange keine Kontrollstrukturen vorkommen, werden die Anweisungen in einem Programm Quelltext vom Compiler der Reihe nach von oben nach unten durchgearbeitet. Die Kontrollstrukturen `if` und `switch` bewirken, dass an einer bestimmten Stelle im Quelltext eine Auswahl zwischen zwei oder mehreren abzuarbeitenden Quelltextteilen getroffen wird. Schleifen bewirken dagegen, dass ein bestimmter Quelltextteil **gar nicht, einmal oder mehrmals wiederholt** wird.

Folgendes Programm ist uns gegeben:

```
System.out.println(„1 zum Quadrat ist “ + (1 * 1));
System.out.println(„2 zum Quadrat ist “ + (2 * 2));
System.out.println(„3 zum Quadrat ist “ + (3 * 3));
System.out.println(„4 zum Quadrat ist “ + (4 * 4));
```

Wie kannst du dieses Programm vereinfachen? Du möchtest alle Quadrate der Zahlen zwischen 1 und 100 ausgeben. Wie würdest du vorgehen?

Zur Programmierung von Schleifen stehen in Java folgende Anweisungen zu Verfügung:

- **do-while-Anweisung** (Fußgesteuerte Schleife) zur Programmierung von **Durchlaufschleifen**. Dabei steht die Bedingung, ob die Schleife durchlaufen werden soll, **nach dem so genannten Schleifenkörper** (den Anweisungen, die wiederholt ausgeführt werden). *Der Schleifenkörper wird also mindestens einmal durchlaufen*. Im Gegensatz zur kopfgesteuerten Schleife wird der Anweisungsblock hier mindestens einmal durchlaufen, weil die Bedingungsprüfung erst im Anschluss an den Anweisungsblock stattfindet.
- **While-Anweisung** (Kopfgesteuerte Schleife) zur Programmierung von **Abweisschleifen**. Dabei steht die Bedingung, ob die Schleife durchlaufen werden soll, **vor dem Schleifenkörper**. *Der Schleifenkörper muss nicht zwangsläufig durchlaufen*. Der Anweisungsblock wird so lange durchlaufen, wie die Bedingung zutrifft.
- **for-Anweisung** (Zählergesteuerte Schleife) zur Programmierung von **Zählschleifen**. Diese Art der Schleife wird stets bevorzugt, wenn zur Berechnung ein Wert durchgezählt wird. Ein typisches Beispiel dafür ist Summieren von aufeinanderfolgenden Zahlen. Die Anzahl der Schleifendurchläufe wird durch eine Zählvariable festgelegt. Im Schleifenkopf werden der Startwert der Zählvariablen, der Endwert und die Veränderung der Zählvariablen (Schrittweite; Iterationsschritte) nach jedem Schleifendurchlauf angegeben.

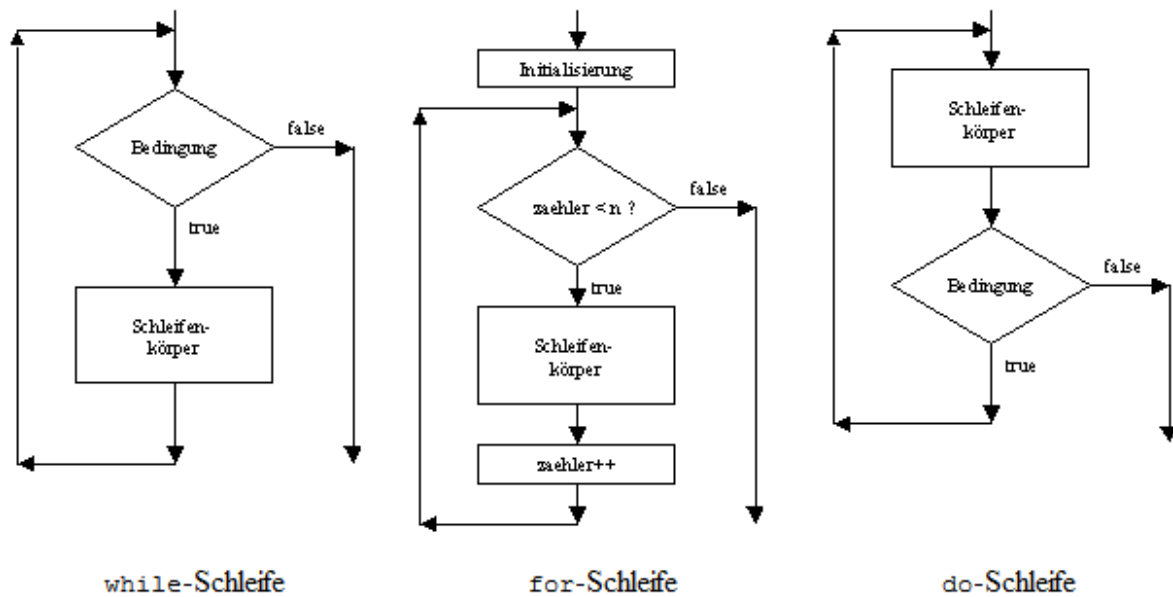


Abbildung 49 Flussdiagramm zu den Schleifen

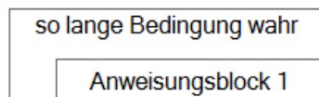


Abbildung 50:  
Struktogramm einer  
kopfgesteuerten Schleife

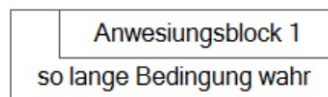


Abbildung 51:  
Struktogramm einer  
Fußgesteuerten Schleife

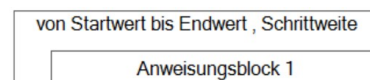


Abbildung 52:  
Struktogramm einer  
Zählergesteuerten Schleife

## Kapitel 5.1: Die for-Schleife

Eine **for-Schleife** benötigt eine Variable, die zu Beginn mit einem Startwert initialisiert wird und führt den nachfolgenden Anweisungsblock solange aus, erhöht oder verringert dabei die Variable um eine konstante Schrittgröße, bis die Bedingung nicht mehr erfüllt ist:

```
for (<Variable> = <Startwert>; <Bedingung>; <Variable> = <Variable> + <Schrittweite>)
{
    <Anweisung>;
    <Anweisung>;
}
```

Abbildung 53

z. B. alle Quadratzahlen für die Werte 1 bis 100 kann man mit einer for-Schleife so ausgeben:

```
for (int i = 1; i <= 100; i = i + 1)
{
    System.out.println (i + "zum Quadrat ist " + (i * i));
}
```

Abbildung 54

Mit **int i=1** wird ein Startwert vorgegeben. Die Schleife führt die Anweisungen innerhalb der geschweiften Klammern solange aus und erhöht jedes Mal i um 1 bis die Bedingung  $i \leq 100$  nicht mehr erfüllt ist.

Eine **for-Schleife** kann stets durch eine **while-Schleife** ersetzt werden, jedoch wird das Programm bei Verwendung einer **for-Schleife** wesentlich übersichtlicher.

## Kapitel 5.2: Die while-Schleife

Manchmal ist es nicht klar, wie viele Schleifendurchläufe benötigt werden, um ein Ergebnis zu erhalten. In dem Fall kann man sogenannte while-Schleife anwenden:

Wiederhole die Anweisungen solange, bis eine Bedingung erfüllt ist.

```
while (<Bedingung>) {
    <Anweisung>;
}
```

Abbildung 56

Bei einer for-Schleifen handelt es sich also um eine fest vorgegebene Anzahl von Wiederholungen. Bei einer while-Schleifen dagegen wird die Anzahl der Wiederholungen von einer Bedingung abhängig gemacht.

Der Schleifenkörper einer **while-Schleife** wird gar nicht erst ausgeführt, wenn die Laufbedingung gleich zu Beginn bereits nicht erfüllt ist. Manchmal ist es aber erwünscht, den Schleifenkörper mindestens einmal

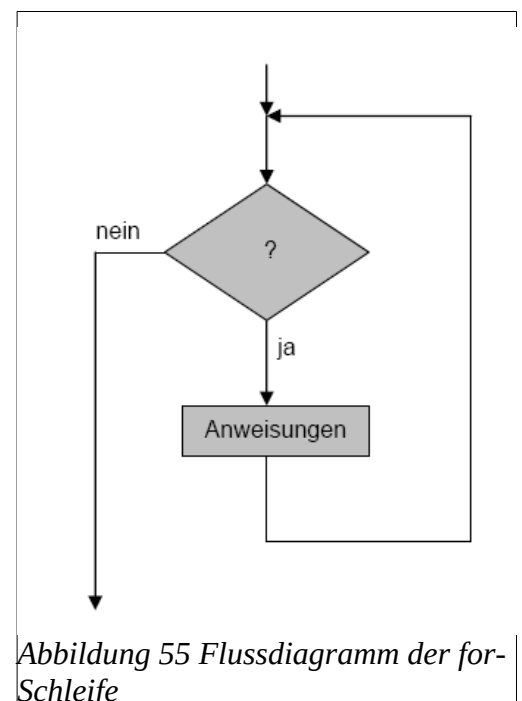


Abbildung 55 Flussdiagramm der for-Schleife

ausführen zu lassen. Anders ausgedrückt: oft ist es sinnvoll, die Bedingung für den Schleifendurchlauf erst nach dem Schleifenkörper zu prüfen. Dies geschieht bei der **do-while-Schleife**.

Quadratzahlenbeispiel:

Die Schleife wird solange ausgeführt, bis die Bedingung hinter while nicht mehr erfüllt ist.

```
int i = 1;
while (i <= 100){
    System.out.println (i + "zum Quadrat ist " + (i * i));
    i = i + 1;
}
```

Abbildung 57

### Kapitel 5.3: Schleifen mit do while

Wie bereits erwähnt wurde, wird der Schleifenkörper einer **while-Schleife** gar nicht erst ausgeführt, wenn die Laufbedingung gleich zu Beginn bereits nicht erfüllt ist. Manchmal ist es aber erwünscht, den Schleifenkörper mindestens einmal ausführen zu lassen. Anders ausgedrückt: oft ist es sinnvoll, die Bedingung für den Schleifendurchlauf erst nach dem Schleifenkörper zu prüfen. Dies geschieht bei der **do-while-Schleife** - zuerst wird den Anweisungsblock einmal ausgeführt und anschließend wird die Bedingung geprüft.

```
do{
    <Anweisung>;
} while (<Bedingung>);
```

Abbildung 59

Beispiel:

```
int i=0;
do {
    System.out.println ("Wert von i: " + i);
    i=i + 1;
} while (i < 5);
```

Abbildung 60

Programmausgabe:

```
Wert von i: 1
Wert von i: 2
Wert von i: 3
Wert von i: 4
Wert von i: 5
```

Die while-Bedingung wird geändert:

```
int i=0;
do{
    System.out.println ("Wert von i: " + i);
} while (i < 0);
```

Abbildung 61

Da die Überprüfung der Bedingung erst nach der ersten Ausführung stattfindet, sieht die Programmausgabe bei dieser Bedingung so aus:

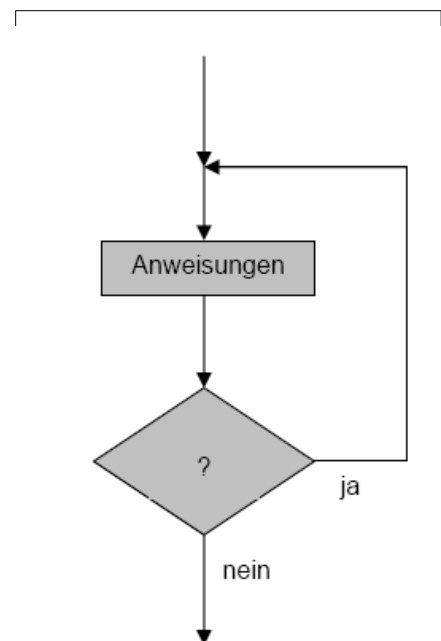


Abbildung 58: Das Flussdiagramm der Do-While-Schleife



## Kapitel 5.4: Schleifensteuerung durch Laufbedingungen

Bei der Schleifensteuerung in Java handelt es sich stets um Laufbedingungen

Beispiel einer for-Anweisung in Java, bei der i von 1 bis 10 hochgezählt wird:

```
for (i = 1; i <= 10; i++)
```

Die Schleife wird 11-mal durchlaufen - solange eine Bedingung erfüllt ist.

## Kapitel 5.5: Terminierung einer Schleife

Bei Schleifen ist darauf zu achten, dass diese stets terminieren; mit anderen Worten: es muss sichergestellt werden, dass die Schleife nicht zur so genannten Endlosschleife wird.

Bei manchen Algorithmen kann es vorkommen, dass diese für bestimmte Parameter nicht konvergieren. Dies geschieht z.B. dann, wenn es keine Lösung für die angegebenen Parameter gibt. In dem Fall würde die Schleifenberechnung nicht beendet werden. Bei solchen Algorithmen sollte eine maximale Anzahl an Schleifendurchläufen vorgegeben werden.

### Beispiel:

Dieses Beispiel zeichnet endlos den Satz "Informatik mit Java ist spitze!" auf den Bildschirm. Endlosschleifen können auch gewünscht sein, z.B. dann, wenn das Programm ständig laufen soll (z.B. bei einer Alarmanlage). Wir werden nun zu jeder Schleifenart ein Beispiel betrachten, um die Unterschiede zwischen den Modellen zu verdeutlichen.

```
while (true)
{
    g.drawString("Informatik mit Java ist spitze!", 50, y);
    y = y + 15;
}
```

Abbildung 62

## Kapitel 5.6: Laufbedingung (Terminierung) bei Verwendung von Gleitkommazahlen

Wenn in der Laufbedingung einer Schleife eine Gleitkommazahl steht, sollte ein Vergleich mit dem Operator == (z.B. a == 0) vermieden werden. Hintergrund: Es kann bei der Berechnung (z.B. aufgrund von Rundungsfehlern) ein Wert herauskommen, der annähernd Null, aber nicht gleich Null ist (z.B. 0.000000001). Die Zahl Null wird eventuell nie erreicht und die Schleife terminiert deshalb nicht. Dieses Problem lässt sich umgehen, wenn man als Vergleichsoperatoren stets „<=“ oder „>=“ verwendet.

## Kapitel 5.7: Schleifen ohne Schleifenkörper

Schleifen können auch ohne Schleifenkörper verwendet werden. In diesem Fall steht nach der Schleifenanweisung ein Semikolon.

**Beispiel:** Um eine Verzögerung in einem Programmablauf zu erreichen, könnte man eine Variable von 0 bis 100000 zählen lassen, ohne dass dabei irgendetwas ausgeführt wird. Die Anweisung könnte dann z.B. so aussehen:

```
for (long i = 0; i < 100000; i++);
```

Im folgenden Beispiel für eine Schleife ohne Schleifenkörper wird gezeigt, dass im Kopf einer **while-Schleife** auch Berechnungen möglich sind.

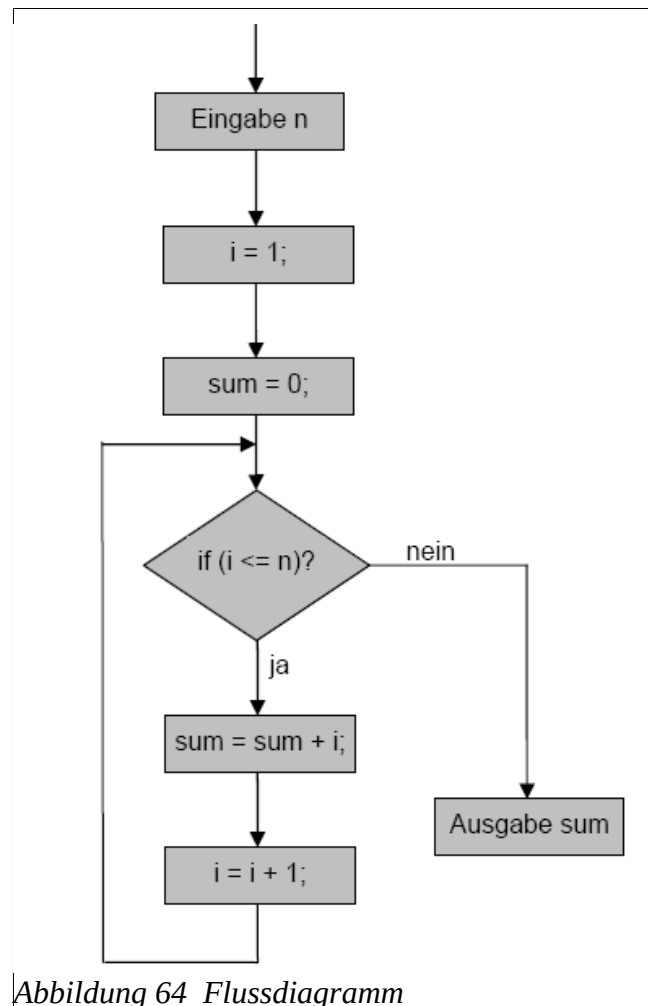
```
int i, j;  
i = 100;  
j = 200;  
while (++i < --j);  
System.out.println("Mittelwert gleich "+i);  
Abbildung 63
```

**Beispiel für eine for-Schleife:** Es sollen alle positiven ganzen Zahlen bis einschließlich n summiert werden. Die Eingabe von n = 0 sei erlaubt, die Eingabe negativer Zahlen für n ist nicht erlaubt.

```
//for-Schleife
int n, sum;
n = 3;
sum = 0;

for (int i = 0; i <= n; i++)
    sum = sum + i;
System.out.println(sum);
```

Abbildung 65



In der folgenden Belegungstabelle werden die Variableninhalte bei der Berechnung für n = 3 dargestellt. Vor Beginn des 4. Schleifendurchlaufs wird das i auf 4 hochgezählt. Nach dem 4. Schleifendurchlauf abgearbeitet wurde wird zuerst geprüft, ob „i<=n“ ist. Da i auf 4 hochgezählt wurde wird die Prüfung „i<=n“ ein nein (false) liefern. Also wird der 5. Schleifendurchlauf nicht durchgeführt.

	i	sum	n	i<=n?
Vor dem ersten Schleifendurchlauf	0	0	3	ja
Erster Schleifendurchlauf mit i=	0	0	3	ja
Zweiter Schleifendurchlauf mit i=	1	1	3	ja
3. Schleifendurchlauf mit i=	2	3	3	ja
4. Schleifendurchlauf mit i=	3	6	3	ja
5. Schleifendurchlauf mit i=	4	--	3	nein

Abbildung 66: Belegungstabelle

In der folgenden Belegungstabelle werden die Variableninhalte bei der Berechnung für n = 0 dargestellt:

	sum	i	n	i <= n?	weiter mit ...
nach Initialisierung	0	1	0	nein	Ausgabe sum

**Beispiel für eine While-Schleife:** Es sollen alle positiven ganzen Zahlen bis einschließlich n summiert werden. Die Eingabe von n = 0 sei erlaubt, die Eingabe negativer Zahlen für n ist nicht erlaubt.

```
//while-Schleife
int i, n, sum;
n = 3;
i = 0;
sum = 0;

while (i <= n) {
    sum = sum + i;
    i++;
}
System.out.println(sum);
```

Abbildung 68

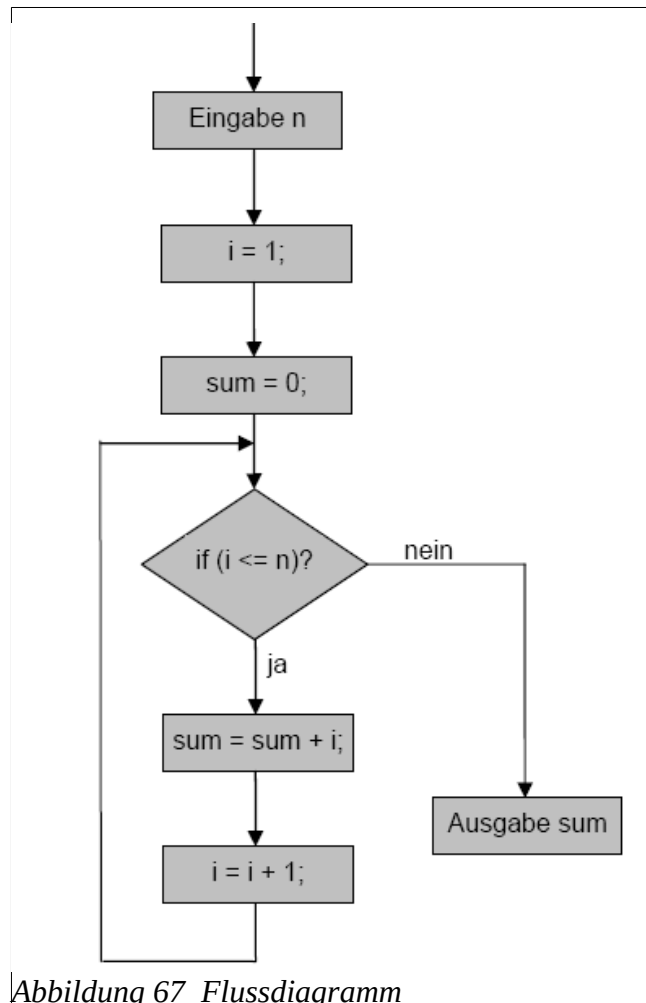


Abbildung 67 Flussdiagramm

In der folgenden Belegungstabelle werden die Variableninhalte bei der Berechnung für n = 3 dargestellt. Beachten Sie, dass bevor die Anweisungen in der Schleife abgearbeitet werden geprüft wird, ob „i<=n“ ist. Wird die Schleife mit i=3 durchlaufen, dann wird am Ende der Schleife i auf 4 gesetzt. Die Prüfung „i<=n“ liefert ein nein (false) und somit werden die Anweisungen in der Schleife nicht noch einmal abgearbeitet. Also wird der 5. Schleifendurchlauf nicht durchgeführt.

	i	sum	n	Die Prüfung „i<=n“ liefert
Vor dem ersten Schleifendurchlauf	0	0	3	ja
Erster Schleifendurchlauf mit i=	0	0	3	ja
Zweiter Schleifendurchlauf mit i=	1	1	3	ja
3. Schleifendurchlauf mit i=	2	3	3	ja
4. Schleifendurchlauf mit i=	3	6	3	ja
5. Schleifendurchlauf mit i=	4	6	3	nein

Abbildung 69: Belegungstabelle

In der folgenden Belegungstabelle werden die Variableninhalte bei der Berechnung für n = 0 dargestellt:

Beachten Sie, dass bevor die Anweisungen in der Schleife

	i	sum	n	Die Prüfung „i<=n“ liefert
Vor dem ersten Schleifendurchlauf	0	0	0	ja
Erster Schleifendurchlauf mit i=	0	0	0	ja
Zweiter Schleifendurchlauf mit i=	1	1	0	nein

Abbildung 70: Belegungstabelle

abgearbeitet werden geprüft wird, ob „ $i \leq n$ “ ist. Wird die Schleife mit  $i=0$  durchlaufen, dann wird am Ende der Schleife  $i$  auf 1 gesetzt. Die Prüfung „ $i \leq n$ “ liefert ein nein (false) und somit werden die Anweisungen in der Schleife nicht noch einmal abgearbeitet. Also wird der zweite Schleifendurchlauf nicht durchgeführt.

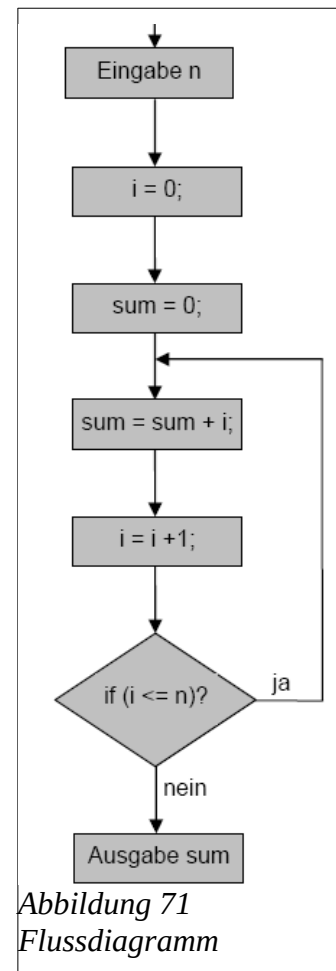
### //do-while-Schleife

```
int i, n, sum;  
n = 3;  
i = 0;  
sum = 0;
```

```
do {  
    sum = sum + i;  
    i++;  
} while (i <= n);
```

```
System.out.println(sum);
```

Abbildung 72 do-while-Schleife



In der folgenden Belegungstabelle werden die Variableninhalte bei der Berechnung für  $n = 3$  dargestellt.

Beachten Sie, dass beim 4. Schleifendurchlauf vor der Prüfung „ $i \leq n$ “ das  $i$  auf 4 hochgezählt worden ist. Deshalb ergibt die Prüfung am Ende des Schleifenrumpfes ein „nein“. Es gibt dann kein weiteren Schleifendurchlauf.

	i	sum	n	Die Prüfung „ $i \leq 4$ “ liefert
Vor dem ersten Schleifendurchlauf	0	0	3	ja
Erster Schleifendurchlauf mit $i=$	0	0	3	ja
Zweiter Schleifendurchlauf mit $i=$	1	1	3	ja
3. Schleifendurchlauf mit $i=$	2	3	3	ja
4. Schleifendurchlauf mit $i=$	3	6	3	nein
5. Schleifendurchlauf mit $i=$	4	6	3	nein

Abbildung 73: Belegungstabelle

In der folgenden Belegungstabelle werden die Variableninhalte bei der Berechnung für  $n = 0$  dargestellt.

Beachten Sie, dass schon beim ersten Durchlauf ( $i=0$ ) das „ $i$ “ vor der Prüfung „ $i \leq n$ “ auf 1 hochgezählt worden ist und damit gibt es kein weiteren Durchlauf.

	i	sum	n	$i \leq n$ ?
Vor dem ersten Schleifendurchlauf	0	0	0	ja
Erster Schleifendurchlauf mit $i=$	0	0	0	nein
Zweiter Schleifendurchlauf mit $i=$	1	1	0	nein

Abbildung 74: Belegungstabelle

**Aufgabe 45:** Erklären Sie, wie lange eine While-Schleife ausgeführt wird.

**Aufgabe 46:** Nehmen Sie Stellung zur folgender Aussage: „Der Schleifenkörper einer **while-Schleife** wird auch dann ausgeführt, wenn die Laufbedingung gleich zu Beginn bereits erfüllt ist.“

**Aufgabe 47:** Erklären Sie, was man bei der Implementierung einer Schleife immer beachten muss.

**Aufgabe 48:** Erklären Sie, welches Problem auftreten kann, wenn eine Schleife implementiert wird, die mit Gleitkommazahl arbeitet.

**Aufgabe 49:** Die Belegungstabelle in der Abbildung 76 zu der for-Schleife in der Abbildung 75 soll ausgefüllt werden.

```
Int x = 9;
for (var i = 0; i <= 4; i++)
{
    x = x*i*2+1;
}
```

Abbildung 75

	i	x	Die Prüfung „i<=4“ liefert
Vor dem ersten Schleifendurchlauf			
Erster Schleifendurchlauf mit i=			
Zweiter Schleifendurchlauf mit i=			
3. Schleifendurchlauf mit i=			
4. Schleifendurchlauf mit i=			
5. Schleifendurchlauf mit i=			
6. Schleifendurchlauf mit i=			
7 Schleifendurchlauf mit i=			

Abbildung 76: Belegungstabelle

**Aufgabe 50:** Die Belegungstabelle in der Abbildung 78 zu der for-Schleife in der Abbildung 77 soll ausgefüllt werden.

```
var x = 9;
var summe=0;
var j=12;

for (var i = 0; i <= 4; i++)
{
    summe = summe + i;
    if (j>12)
    {
        j=j+1;
        x=x+summe+j;
    }else
    {
        j=j+1;
        x=x+summe*j;
    }
}
```

Abbildung 77

	i	x	summe	j	Die Prüfung „i<=4“ liefert
Vor dem ersten Schleifendurchlauf					
Erster Schleifendurchlauf mit i=					
Zweiter Schleifendurchlauf mit i=					
3. Schleifendurchlauf mit i=					
4. Schleifendurchlauf mit i=					
5. Schleifendurchlauf mit i=					
6. Schleifendurchlauf mit i=					
7. Schleifendurchlauf mit i=					

Abbildung 78: Belegungstabelle

**Aufgabe 52:** Betrachten Sie das folgende Programm und übersetze den unterlegten Teil in Umgangssprache. Kommentieren Sie jede Zeile. Geben Sie das Struktogramm zum Quelltext an.

```
public class BedeutungForSchleife {
    public static void main (String [] arguments){
        int i;
        for(i=0; i<10; i++){
            System.out.println(i);
        }
    }
}
```

Abbildung 79

**Aufgabe 53:** Erstelle ein Programm, das alle Zahlen von 1 bis 100 aufaddiert. Das Ergebnis jeder Addition soll ausgegeben werden.

**Aufgabe 54:** Erstelle ein Programm, das für alle Zahlen zwischen 1 und 100 testet, ob sie jeweils durch die angegebene Zahl ganzzahlig teilbar sind und geben Sie diese Information auf der Konsole aus.

- a) durch 3 teilbar sind.
- b) nicht durch 5 aber durch 4 teilbar sind.
- c) Primzahlen sind.

**Aufgabe 55:** Was passiert in dem folgenden Programm? Übersetze den unterlegten Teil in Umgangssprache! Kommentieren Sie jede Zeile

```
public class WhileSchleife{
    public static void main (String [] arguments){
        int x;
        x = 10;
        while (x > 0){
            System.out.println(x);
            x = x-3;
        }
    }
}
```

Abbildung 80

**Aufgabe 56:** Erstellen Sie zu folgenden Ablauf einer Kopfgesteuerten Schleife ein Struktogramm und ein Flussdiagramm: Ein Läufer läuft Runden in einem Stadion. Sein Trainingsprogramm besagt, dass er keine weitere Runde laufen soll, wenn sein Puls den Wert von 160 überschreitet.

Implementieren Sie die Schleife in Java mit einer entsprechenden Ausgabe.

**Aufgabe 57:** Folgendes Struktogramm (Abbildung 81) ist gegeben. Geben Sie das dazugehörige Flussdiagramm an und Geben sie den Ablauf in eigenen Worten wieder (Ins Heft Notieren). Geben Sie an, von welcher Art Schleife es sich hierbei handelt.

**Aufgabe 58:** Folgender Ablauf soll mit Hilfe einer Schleife implementiert werden: Bei einem Würfelspiel wird mit einem Würfel so lange gewürfelt, bis eine 6 fällt. Die Anzahl der Würfe wird gezählt. Wenn eine 6 gefallen ist, wird die Anzahl der

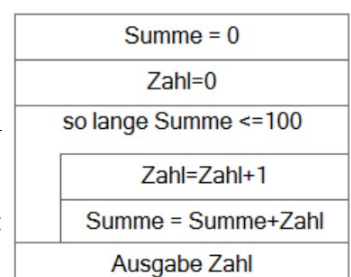


Abbildung 81

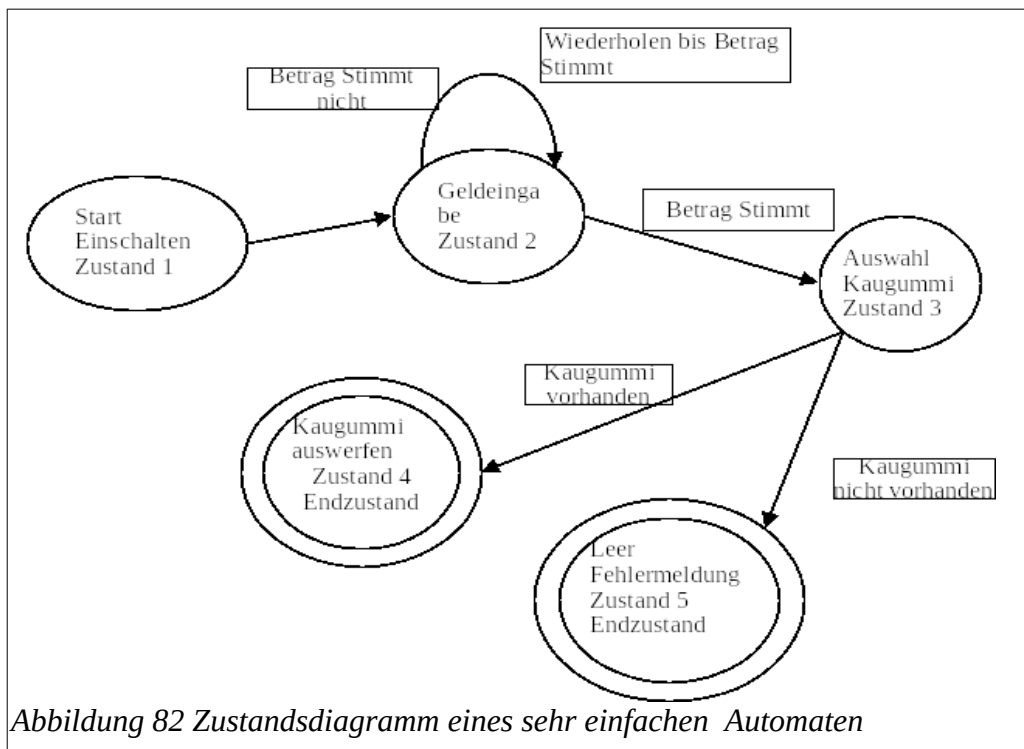


Würfe ausgegeben.

Geben Sie das dazugehörige Flussdiagramm (for-Schleife; While-Schleife; do-While-Schleife) und Struktogramm (for-Schleife; While-Schleife; do-While-Schleife) an. Implementieren Sie die Schleifen in Java (mit einer entsprechen Ausgabe!).

### Kapitel 5.7.1: Aufgabe: Projekt Automat

**Aufgabe 59:** In der Abbildung Abbildung 82 sehen Sie ein Zustandsdiagramm eines Automaten. In der ersten Version ihres Automaten soll der Automat ein Kaugummi ausgeben können (es gibt nur eine Auswahl). Eine Eingabe über die Konsole soll ebenso implementiert werden. Erstellen Sie für das Projekt ein eigenes Verzeichnis mit Namen „Automat\_v1“.



Um Ihnen den Einstieg in das Projekt zu erleichtern gehe ich mit Ihnen die ersten Schritte durch. Erstellen Sie eine Klasse mit Namen „Automatanwendung“. Über die Zeile „public class Automatanwendung{“ wird die Scanner-Klasse importiert („import java.util.Scanner;“) In der Klasse „Automatanwendung“, in der main-Methode, sind folgende Variablen definiert:

```
Scanner Eingabe = new Scanner (System.in);
public double summeMuenzeinwurf=0; //In euro
public String auswahl="Kaugummi"; //Auswahl steht fest
public double muenzeinwurf=1; //In euro
public String auswahlKaugummi = "Kaugummi"; //Nur eineAuswahl
public int AnzahlKaugummi = 10; //Nur 10 Kaugummis sind im Automat vorhanden
public double preisKaugummi=3; //inEuros
```

Abbildung 83

Als nächstes verwenden Sie eine while-Schleife, die so lange läuft so lange „summeMuenzeinwurf“ nicht gleich „preisKaugummi“ ist. In der Schleife bilden Sie die Differenz zwischen „preisKaugummi“ und „summeMuenzeinwurf“. Somit soll der User darauf hingewiesen, dass er noch mehr münzen einwerfen muss. Die Münzeingabe muss zu der Variablen „summeMuenzeinwurf“ addiert werden. Nach dem verlassen der Schleife geben Sie die Meldung „Sie können Ihre Ware entnehmen“ auf die Konsole ausgeben. Vergessen Sie nicht die Variable „AnzahlKaugummi“ um eins zu erniedrigen, da ja jetzt nur noch 9 Kaugummis vorhanden sind.

Damit ist Ihre erste Version des Automaten fertig.

- a) Erstellen Sie für eine Erweiterung Ihres Automaten ein eigenes Verzeichnis mit Namen „Automat\_v2“. Speichern Sie alle Dateien in dieses Verzeichnis. So bleibt ihre erste Version Ihres Automaten erhalten.

Jetzt sollen Sie Ihr Automat um weitere Auswahlmöglichkeiten, wie Kekse oder Schnittchen, oder einen Getränk oder einen Riegel, erweitern.

Jedes Produkt muss gezählt werden, d.H. z.B., dass der Riegel „Körni“ eine Variable „AnzahlKoerni“ erhält: „public int AnzahlKoerni = 10;“  
Natürlich braucht jedes Produkt auch ein Preis.

Bei jeder Ausgabe eines „Körnis“ wird die Variable „AnzahlKoerni“ um eins erniedrigt.

**Aufgabe 60:** Die nächste Erweiterung des Automaten. Erstellen Sie für das Projekt ein eigenes Verzeichnis mit Namen „Automat\_v2“ und kopieren Sie alle Dateien der vorherigen Automatenversion in dieses Verzeichnis.

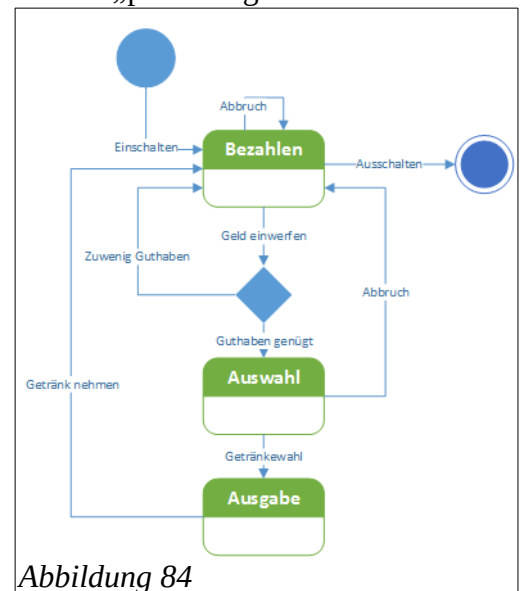
In dieser Automatenversion, soll der Automat ...

- ... mehrere Produkte anbieten.
- ... beliebige Geldeingaben akzeptieren, also 5ct, 10ct, ....
- ... sich das Datum des Verkaufs, das Produkt welches verkauft wurde und wie viel es gekostet hat merken
- ... drei Arrayvariablen oder eine drei dimensionale Arrayvariable verwenden um sich die Daten zu merken. Arrayvariablen werden erst im nächsten Kapitel behandelt. Aber an dieser Stelle können Sie sehen, wo für Arrays benötigt werden.
- ... auf Anfrage ermitteln wann der größte Umsatz erzielt worden ist, wann der niedrigste Umsatz erzielt worden ist und wie viel Umsatz in welchen Zeitraum insgesamt erzielt worden ist.

Tipp für das Datum:

```
import java.time.LocalDate; // import the LocalDate class
LocalDate myObj = LocalDate.now(); // Create a date object
```

Beispiele: [https://www.w3schools.com/java/java\\_date.asp](https://www.w3schools.com/java/java_date.asp) und <https://mkyong.com/java/java-how-to-get-current-date-time-date-and-calender/>



Wenn man nicht nur einen int-Wert, sondern viele davon verwalten möchte, könnte man dies in etwa so bewerkstelligen, wie im der Abbildung 85 zu sehen ist.

Diese Methode ist sehr aufwändig und nicht besonders elegant. Einfacher wäre, wenn man sagen könnte, dass man  $k$  verschiedene *int*-Werte hat und diese dann über einen Index anspricht. Genau das nennt man eine Liste oder ein Feld (Array).

Ein Feld ist eine Datenstruktur. In einer Datenstruktur werden auf eine bestimmte Art und Weise Daten gespeichert. Die Variablen sind auch Datenstrukturen, in einer Variablen wird aber ein Wert von einem bestimmten Datentyp gespeichert. In einem Feld dagegen werden mehrere Daten vom gleichen Datentyp gespeichert.

```
int a, b, c, d, e, f;
a = 0;
b = 1;
c = 2;
d = 3;
e = 4;
f = 5;
```

Abbildung 85

Man kann sich ein Feld wie eine Liste vorstellen, wo eine Information nach der anderen steht. Es gibt aber auch komplexere Felder: die so genannten zweidimensionalen Felder. Diese kann man sich am besten als Tabellen vorstellen

*Beispiele:*

a) Wenn die durchschnittliche Höchsttemperatur in einem Monat bestimmt werden soll, ist es ungeschickt 31 double-Variablen temp1, temp2, ... , temp31 zu verwenden. Geschickter ist ein Feld, da man dessen Elemente sehr geschickt in Schleifen bearbeiten kann.

b) Ein typisches Beispiel für Felder in der Mathematik sind Vektoren.

## Kapitel 6.1: Deklaration und Zuweisung

Zunächst muss das Feld deklariert werden. In der Deklaration des Feldes, wird mitgeteilt welcher Variablen das Feld zugewiesen werden soll und von welchem Datentyp die Werte des Feldes sind. Um ein  $k$ -elementigen *int*-Feld zu erzeugen, schreibt man:

```
<Datentyp> [ ] <Name>;
<Name> = new <Datentyp> [k];
```

Oder in einer Zeile zusammengefasst:

```
<Datentyp> [ ] <Name> = new <Datentyp> [k];
```

Mit dem Befehl *new* wird Speicher für das Feld bereitgestellt.  $k$  ist Anzahl der Speicherzellen, die in dem Feld zur Verfügung stehen sollen. Zu beachten ist, dass das erste Element eines Feldes mit dem Index 0 und das letzte der  $k$  Element mit dem Index  $k - 1$  angesprochen werden. Die Speicherzellen des Feldes werden also von 0 bis  $k-1$  durchnummeriert

**Indizes**  
**Elemente**

0	1	2	3	4	5	6	7	8	9	10	11	12	13
2	3	9	0	11	4	3	5	6	8	1	9	12	7

Visualisierung eines Feldes mit 14-Elementen. Es ist darauf zu achten, dass das erste Element immer mit dem Index 0 startet.

Nummer der Schublade

Inhalt der Schublade

Nachdem das Feld erzeugt ist, ist jede Speicherzelle des Feldes bereits mit einem Standardwert belegt (initialisiert) worden. Ein Feld mit dem Datentyp Integer wird mit dem Wert 0 initialisiert.

Ein Arrayvariable kann man sich vorstellen, wie ein Schrank mit vielen Schubladen. Die Schubladen werden durchnummeriert (Index oder Position). In den Schubladen werden die Werte reingelegt (Elemente).

Beispiel.: Definition eines Feldes aus 31 Double-Elementen mit dem Variablennamen „Temperatur“:

double[] Temperatur; Temperatur=new double[31];	Oder	double[] Temperatur=new double[31];
--	------	-------------------------------------

Wenn man das Feld gleich mit von 0 verschiedenen Werten initialisieren möchte, geschieht dies durch z.B.:

int [ ] gerade = {2,4,6,8,10,12};	Oder	int [ ] gerade = new int [2]; gerade [0] = 2; gerade [1] = 4;
-----------------------------------	------	---

String [ ] namen = {"Michael","Maike","Jörg"};	Oder	String [ ] namen = new String [2]; namen [0] ="Michael"; namen [1] = "Maike";
--	------	---

## Kapitel 6.2: Zugriff auf Elemente eines Feldes

Auf die Elemente eines Feldes kann man mit „**Variablenname [Index]**“ zugreifen. Wenn Index außerhalb von Feld-Anfang und Feld-Ende liegt, erhält man in der Regel eine Fehlermeldung.

Beispiel:

Mit Hilfe von „*gerade [3]*“ liest man die Zahl „8“ aus dem obigen Feld, „*namen [0]*“ liefert „Michael“ zurück. Nach der Anweisung „*zahl = gerade[4];*“ hat die Variable zahl vom Typ „int“ den Wert 10. Aus der Schublade mit der Nummer „4“ wird der Wert „10“ raus genommen und in die Variable „zahl“ kopiert. Der wert „10“ bleibt dennoch im Array erhalten.

Einer Komponente eines Feldes weist man den Wert durch „*Temperatur[3]=25.4;*“ zu. Dadurch wird der vorherige Wert an der Position mit dem Index 3 überschrieben. In die Schublade mit der Nummer „3“ kommt jetzt der Wert „25,4“ rein. Der alte Wert in der Schublade mit der Nummer „3“ geht verloren.

Beispiel: In der Abbildung 86 wird ein Array mit Zufallszahlen befüllt. Dann wird der Inhalt des Array ausgegeben und dann die Summe der Zahlen im Array berechnet.

Im Beispielprogramm (Siehe Abbildung 86) wird eine wichtige Technik beim Umgang mit Feldern verwendet.

Felder sind in Java Objekte. Deshalb kann man alternativ dazu auch die Eigenschaft „**length**“ des Feldes nutzen, d.h. (2) und (3) würden so aussehen:

```
for(int i=0; i<zahlen.length; i++){ ... }
```

```
public class Summe{
    public static void main(String[] args){
        int anzahl=5; // Anzahl der zu speichernde Elemente festlegen
        double[] zahlen; // Feld deklarieren
        zahlen=new double[anzahl]; // Feld erzeugen

        for(int i=0;i<anzahl;i++){ // (2) Zahlen in das Array Schreiben
            zahlen[i]= Math.random(); //erzeugt Zufallszahlen
        }

        for(int i=0;i<anzahl;i++){ // (3) Zahlen auslesen
            //Werte aus dem Array nehmen und anschließend anzeigen.
            int schubladeninhalt = zahlen[i];
            System.out.println(" schubladeninhalt = "+schubladeninhalt);
        }

        double summe=0; // Zahlen summieren

        for(int i=0;i<anzahl;i++){ // (3)
            summe=summe+zahlen[i];
        }
        System.out.println("Summe = "+summe); // Summe ausgeben
    }
}
```

Abbildung 86

**Aufgabe 61:** Felder deklarieren, anlegen und ausgeben. Programmieren Sie die folgenden Felder!

x	y	z	v
0	0	0	0
1	1	1	1
2	2		2
	3		

12
11
-1

1.3
1.4
-12.3
2.23

'a'
'#'

"Ar"
"ra"
"y"

Lassen Sie in Ihrem Programm folgende Werte am Bildschirm ausgeben:

- Inhalt der Speicherzelle 0 vom Feld x
- Inhalt der Speicherzelle 3 vom Feld y
- Inhalt der Speicherzelle 1 vom Feld z
- Inhalt der Speicherzelle 0 vom Feld v

**Aufgabe 62:** Felder deklarieren, anlegen und ausgeben. Erstellen Sie ein Programm, in dem Sie ein Feld für zwei unterschiedliche Datentypen deklarieren, anlegen und mit Werten füllen. Die beiden Felder sollen eine unterschiedliche Anzahl an Speicherzellen haben. Der Inhalt jedes Feldes soll am Bildschirm ausgegeben werden. Formatieren Sie die Ausgabe so wie in der Abbildung 87 dargestellt ist.

doubleArray ist ein Feld, das 3 double-Werte enthält.

-----  
Position im Array: 0 | Inhalt: 3.1  
Position im Array: 1 | Inhalt: 12.0  
Position im Array: 2 | Inhalt: -3.4

Abbildung 87

**Aufgabe 63:** Schleifen und Felder. Erstellen Sie ein Programm, in dem alle Zahlen von 1 bis 100 in ein geeignetes Array geschrieben werden. Verwenden Sie eine Schleife. Anschließend soll der Inhalt des Arrays mit Hilfe einer while-Schleife ausgegeben werden. Formatieren Sie die Ausgabe so wie in der Abbildung 88 dargestellt ist.

doubleArray ist ein Feld, das 100 int-Werte enthält.

-----  
Position im Array: 0 | Inhalt: 3.1  
Position im Array: 1 | Inhalt: 12.0  
Position im Array: 2 | Inhalt: -3.4

Abbildung 88

**Aufgabe 64:** Ändern Sie das obige Beispielprogramm so ab, dass man beliebig viele Elemente eines Feldes addieren kann. Dazu muss zu Programmbeginn der Benutzer gefragt werden, wie viele Werte er addieren möchte und welche.

**Aufgabe 65:** Erstellen Sie ein neues Programm, das jetzt das Produkt der eingegebenen Elemente eines Feldes bestimmen soll.

**Aufgabe 66:** Erstellen Sie ein neues Programm, das den arithmetischen Mittelwert der eingegebenen Elemente eines Feldes bestimmt.

**Aufgabe 67:** Erstellen Sie ein Programm, das Minimum, Maximum der eingegebenen Elemente eines Feldes und ihre Positionen bestimmt.

**Aufgabe 68:** Notenverwaltung. Erstellen Sie ein Programm zur Verwaltung der Noten einer Schulklasse. Das Programm hat 4 Funktionen.

- Noten der Klausur einlesen lassen, wobei die Klassengröße interaktiv in Erfahrung gebracht wird.
- Notenliste der Klausur ausgeben lassen
- Durchschnittsnote der Klasse ermitteln lassen.
- Programm beenden.

Die Funktionen werden über ein immer wiederkehrendes Menü angesteuert (Tipp: switch und while).

## Kapitel 7: Zweidimensionale Felder (Arrays) in Java

Für verschiedene Anwendungen ist es notwendig, Arrays mit mehreren Dimensionen anzulegen und zu verwenden. Ein Spezialfall mit der Dimension 2 ist die Matrix. Zweidimensionale Arrays sind **Tabellen** zum Speichern von Werten des gleichen Datentyps.

### Beispiel:

- y ist ein zweidimensionales Array für *int*-Werte
- y hat 3 Zeilen und 4 Spalten
- Die Speicherzelle in der Zeile 1 und der Spalte 2 enthält eine 12.

Indizes	y				
	0	1	2	3	
0	1	-9	34	-8	<i>Element [3][0]</i>
1	6	-6	12	0	
2	9	89	0	0	<i>Element [1][2]</i>

### Kapitel 7.1: Deklaration eines zweidimensionalen Arrays

**Datentyp** Name [] [];

An den **zwei** eckigen Klammern wird erkannt, dass es sich um ein zweidimensionales Array handelt.

### Kapitel 7.2: Anlegen eines zweidimensionalen Arrays

Name = **new** **Datentyp** [zeilenanzahl] [spaltenanzahl];

Die Zeilen (Spalten) des Arrays werden von 0 bis zeilenanzahl-1 (spaltenanzahl-1) durchnummeriert.

### Kapitel 7.3: Speichern und Auslesen bei zweidimensionalen Arrays

Jede einzelne Speicherzelle kann mit dem Namen des Arrays, der Nummer ihrer Spalte und der Nummer ihrer Zeile angesprochen werden und zwar **Name [Zeile][Spalte]**, wobei Zeile (Spalte) eine ganze Zahl zwischen 0 und Zeilenanzahl-1 ist.

```

public class ZweidimensionalesArray{
    public static void main (String [] arguments){
        int y[][];           //Deklaration von y als zweidimensionales int-Array
        y = new int [3][4];   //Anlegen von y mit 3 Zeilen und 4 Spalten
        // Speichern der Werte in Zeile 0
        y[0][0] = 1;
        y[0][1] = -9;
        y[0][2] = 34;
        y[0][3] = -8;
        // Speichern der Werte in Zeile 1
        y[1][0] = 6;
        y[1][1] = -6;
        y[1][2] = 12;
        y[1][3] = 0;
        // Speichern der Werte in Zeile 2
        y[2][0] = 9;
        y[2][1] = 89;
        y[2][2] = 0;
        y[2][3] = 0;
        // Ausgabe der verlangten Werte
        System.out.print("y[0] [0] = "); System.out.println(y[0] [0]);
        System.out.print("y[2] [1] = "); System.out.println(y[2] [1]);
        System.out.print("y[1] [3] = "); System.out.println(y[1] [3]);
        System.out.print("y[2] [3] = "); System.out.println(y[2] [3]);
    }
}

```

Abbildung 89

### Aufgabe 69: Stundenplanmanager

Erstellen Sie einen Stundenplanmanager! Bearbeiten Sie die folgende Schritte alle im gleichen Programmquellcode der Reihe nach. Gehen Sie immer erst zum nächsten Schritt über, wenn der aktuelle gemeistert ist.

- Lege ein zweidimensionales Array an, das sich dazu eignet, einen Stundenplan zu speichern (z.B. 5 Spalten 6 Zeilen).
- Programmiere eine entsprechende Ausgabe für den Stundenplan (also für das in 1 angelegte Array). Verwenden Sie Schleifen für die Ausgabe.  
Beispiel:

Aktueller Stundenplan				
Mo	Di	Mi	Do	Fr
M	If	M	E	Ph
M	Sp	M	E	Ph
E	D	Sp	M	E
E	D	Sp	F	E
Ph	F	K	If	K
	F		If	K



c) Erweitere dein Programm so, dass du die Stunden selbst eintragen kannst

Beispiel:

```
** Stunden eintragen **  
Tag 1=Mo    2=Di  3=Mi  4=Do oder 5=Fr  
Stunden 1 – 6  
Welche Stunde soll eingetragen werden?  
Tag: 2  
Stunde: 1  
Fach: If
```

d) Erweitere dein Programm so, dass man die Möglichkeit hat in einem Menü aus folgenden 3 Punkten auszuwählen:

- i. Stundenplan ausgeben
- ii. Stunde eintragen
- iii. Programm beenden

Nach Ausführung des entsprechenden Menüpunktes wird zum Menü zurückgekehrt. Tipp: case - switch-Block und while-Schleife! Verwenden Sie Schleifen für die Ausgabe (Für die Darstellung des Stundenplans).

Beispiel:

```
***** Stundenplanmanager *****  
(1) Stundenplan ausgeben  
(2) Stunde eintragen  
(3) Programm beenden  
Bitte wähle einen Menüpunkt: 2  
  
** Stunden eintragen **  
Tage 1=Mo    2=Di  3=Mi  4=Do oder 5=Fr  
Stunden 1 – 6  
Welche Stunde soll eingetragen werden?  
Tag: 2  
Stunde: 1  
Fach: If  
Abbildung 90
```

## Kapitel 8: Methoden bzw. Funktionen in Java

### Kapitel 8.1: Was sind Methoden?

Funktionen sind eigenständige Programmteile, die vom Hauptprogramm beliebig oft aufgerufen und abgearbeitet werden können. Funktionen beinhalten Anweisungen, die innerhalb des Programms oder innerhalb eines Projekts mit mehreren Programmdateien mehrmals benötigt werden. Anstatt die Anweisungen mehrfach im Programm zu codieren, wird die entsprechende Funktion einmalig definiert und an den gewünschten Stellen aufgerufen, um die Anweisungen der Funktion dort auszuführen. Bei einem Aufruf der Funktion werden alle in ihr zusammengefassten Befehle schrittweise ausgeführt, so als wären sie jeweils einzeln aufgerufen worden. In Java gibt es genau genommen keine Funktionen, sondern nur Methoden. Eine Methode ist nichts anderes als eine Funktion, die zu einer Klasse gehört.

### Kapitel 8.2: Vorteile von Methoden

- Immer wiederkehrende Abläufe werden nur einmal beschrieben und können danach beliebig oft ausgeführt werden.
- Der Programmcode wird mithilfe von Funktionen strukturiert und ist dadurch einfacher zu pflegen.
- Änderungen am Programm lassen sich schneller und einfacher durchführen, da eine Änderung nur einer bestimmten Stelle nötig ist.
- Die Struktur des Programms lässt sich leichter nachvollziehen, da der Quellcode übersichtlicher ist.

### Kapitel 8.3: Eine Methode erstellen

Es wurde eine Ausgabe für Zahlen programmiert und die nach jedem Berechnungsschritt ausgeführt werden.

Um Redundanz zu vermeiden und die Programmzeilen nur einmal aufzuschreiben, lagert man diese Zeile in eine Methode aus. Wir nennen diese Methode „*gibAus*“.

```
public class Aufgabe_1 {  
    public static void main(String[] args) {  
        int a=4;  
        System.out.println();  
        System.out.println("*****");  
        System.out.println("*** Wert der Variable ist " +a);  
        System.out.println("*****");  
        System.out.println();  
        a=(a*13)%12;  
        System.out.println();  
        System.out.println("*****");  
        System.out.println("*** Wert der Variable ist " +a);  
        System.out.println("*****");  
        System.out.println();  
        a+= 1000;  
        System.out.println();  
        System.out.println("*****");  
        System.out.println("*** Wert der Variable ist " +a);  
        System.out.println("*****");  
        System.out.println();  
    }  
}
```

Abbildung 91

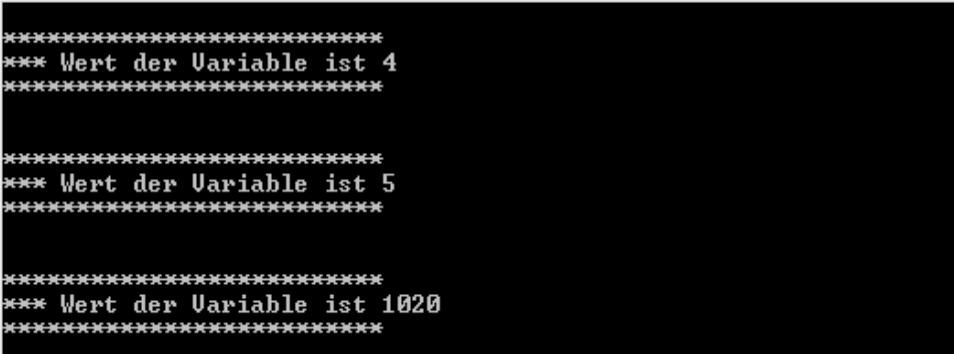
### Kapitel 8.3.1: Die Methode (Funktion) gibAus

Nach dem das Programm gestartet wurde, bekommt man die Ergebnisse, wie sie in der Abbildung 93 dargestellt sind.

Aus der Mathematik ist bekannt, dass Funktionen **ein Ergebnis liefern** – genau ein Ergebnis. Das gilt auch in der Programmierung. Falls, kein Rückgabewert existiert, schreib man bei der Definition der Funktion das Schlüsselwort „**void**“ als Rückgabewert.

```
public class Aufgabe_1_funk {  
    // Funktion gibAus  
    public static void gibAus (int a)  
    {  
        System.out.println();  
        System.out.println("*****");  
        System.out.println("*** Wert der Variable ist " +a);  
        System.out.println("*****");  
        System.out.println();  
    }  
  
    public static void main(String[] args) {  
        int tom=4;           // Startwert für lokales a  
        gibAus(tom);         //Funktion gibAus wird aufgerufen  
        int a=5;  
        a=(a*13)%12; // Variable a wird geändert  
        gibAus(a);           //Funktion gibAuf wird aufgerufen  
        int b=20;  
        b+= 1000;  
        gibAus(b);           //Funktion gibAuf wird aufgerufen  
    }  
}
```

Abbildung 92



```
C:\WINDOWS\system32\cmd.exe  
  
*****  
*** Wert der Variable ist 4  
*****  
  
*****  
*** Wert der Variable ist 5  
*****  
  
*****  
*** Wert der Variable ist 1020  
*****
```

Abbildung 93

### Kapitel 8.4: Deklaration von Methoden (Funktionen)

Hierbei ist

- „public static <Rückgabety>  
<Funktionsname>  
(<Parameterliste>)“ der  
Methoden Kopf.
- <Rückgabety> - der Typ des  
Ergebnisses, das die Funktion  
zurückliefern soll (**int**,  
**double**, **float**, **String** usw.) oder **void**, falls kein Wert zurückgegeben wird.

```
public static <Rückgabety> <Funktionsname> (<Parameterliste>)  
{  
    // Methoden- bzw. Funktionsrumpf: hier den  
    // auszuführenden Code einfügen  
}
```

Abbildung 94

- *<Funktionsname>* - ein Bezeichner, unter dem die Methode von Java erkannt werden soll. Der Funktionsname darf selbstverständlich kein reserviertes Wort sein.
- *<Parameterliste>* - eine Komma separierte Liste von Variablendeklarationen. Die darin aufgeführten Variablen werden formale Parameter oder auch formale Argumente genannt und fungieren als Platzhalter für Werte, die an die Funktion übergeben werden sollen.

Unterprogramme (Methoden; Funktionen), die aus der **main** Methode aufgerufen werden, müssen als **static** deklariert werden.

Wenn wir uns den Funktionskopf etwas genauer ansehen, erkennen wir eine große Ähnlichkeit mit der bereits bekannten Zeile „public static void main (String[] args)“

Ist das Zufall? Nein, natürlich nicht. Tatsächlich ist die Hauptmethode, die wir bislang immer verwendet haben, nichts anderes als eine Methode. Sie hat als Rückgabewert **void**, also liefert sie keinen Wert als Ergebnis. Der Methodename ist „**main**“, und die Parameterliste besteht aus einem Feld von Typ „**String**“, das den Namen „**args**“ trägt.

**Aufgabe 70:** Geben Sie an, welche Vorteile sich ergeben, wenn man Methoden verwendet.

**Aufgabe 71:** Erklären Sie, was mit „<Rückgabotyp>“ gemeint ist.

**Aufgabe 72:** Erklären Sie, was man bei der Namensgebung von Funktionen beachten muss.

**Aufgabe 73:** Erklären Sie, was man unter „<Parameterliste>“ versteht und geben Sie die Funktion der „<Parameterliste>“ an (Wofür braucht man das?).

**Aufgabe 74:** Erklären Sie, wann man das Schlüsselwort „void“ im Funktionskopf angibt.

**Aufgabe 75:** Füllen Sie die Lücken im Text in der Abbildung 95 aus. Verwenden Sie vorgegebenen Begriffe aus der Abbildung 96

Funktionen sind \_\_\_\_\_ Programmteile, die vom \_\_\_\_\_ beliebig \_\_\_\_\_ aufgerufen und \_\_\_\_\_ werden können.

Anstatt die selben \_\_\_\_\_ mehrfach im Programm zu \_\_\_\_\_, wird die entsprechende Methode nur \_\_\_\_\_ definiert und an den gewünschten Stellen \_\_\_\_\_, um die Anweisungen der Methode genau dort \_\_\_\_\_.

Funktionen beinhalten Anweisungen, die innerhalb des Programms oder innerhalb eines \_\_\_\_\_ mit mehreren Programmdateien \_\_\_\_\_ benötigt werden.

In Java gibt es genaugenommen keine \_\_\_\_\_, sondern nur Methoden. Eine Methode ist nichts anderes als eine \_\_\_\_\_, die zu einer Klasse gehört.

Bei einem Aufruf der Funktion werden alle in ihr \_\_\_\_\_ Befehle schrittweise ausgeführt, so als wären sie jeweils \_\_\_\_\_ aufgerufen worden.

Abbildung 95

Implementieren, zusammengefassten, auszuführen, Hauptprogramm, Projekts, Anweisungen , ein mal, mehrmals, eigenständige, oft, aufgerufen, einzeln, abgearbeitet, Funktionen, Funktion

Abbildung 96

**Aufgabe 76:** Geben Sie die jeweiligen Methodenköpfe für folgende Methoden an:

Beschreibung des Methodekopfes	In Java
Eine Methode, die von überall aufrufbar ist, also „public“, die mit dem Bezeichner „static“ gekennzeichnet ist, kein Rückgabetyt erwartet, den Namen „main“ trägt und nur ein Parameter mit Namen „args“ vom Typ „String[]“, also ein Array vom Typ String, enthält.	public static void main(String[] args)
Eine Methode, die von überall aufrufbar ist, die mit dem Bezeichner „static“ gekennzeichnet ist, den Rückgabetyt „boolean“ erwartet, den Namen „pruefe“ trägt und nur ein Parameter mit Namen „wahrenannahmeP“ vom Typ „String[]“, enthält.	
Eine Methode, die von überall aufrufbar ist, den Rückgabetyt „double“ erwartet, den Namen „querschnitt“ trägt und zwei Parameter mit Namen „anzahlP“ bzw. „summeAngeboteP“ vom Typ „int“ bzw. „double“, enthält.	
Eine Methode, die als „private“ deklariert ist, also nicht von überall aufrufbar, den Rückgabetyt „long“ erwartet, den Namen „anzahlDatensaetze“ trägt und drei Parameter mit Namen „selectAnweisungP“, „tabellennameP“ und „minAnzahl“ vom Typ „String“, „String“ und „int“, enthält.	
Eine Methode, die als „public“ deklariert ist, ein Array als Rückgabe vom Typ „long“ erwartet, den Namen „anzahlNachkommastellen“ trägt und drei Arrayparameter mit Namen „selectAnweisungP“, „tabellennameP“ und „minAnzahl“ vom Typ „String“, „String“ und „int“, enthält.	

**Aufgabe 77:** Beschreiben Sie die folgenden Methodenköpfe

In Java	Beschreibung des Methodekopfes
public int gibZufallsZahl(int[] zufallszahlenP)	
public static boolean stockwerk(int raumNummerP, String nameEtageP)	
public double termin(double zeitRaumP, String monatP, int jahresZahlP)	
public int[] statistik(double[] gemittelteZahlP, double[] nichtgemittelteZahlP, int[] teilerP)	

Erstellen Sie zu folgenden Anweisungen eine Methode. Den Namen der Methode können Sie sich aussuchen. Verwenden Sie sprechende Bezeichnungen!

```
public class FlaecheUmfangRechteckBerechnen {

public static void main(String[] args) {
    double breite=4;
    double hoehe=15;

    //was die Methode abarbeiten soll
    double flaeche= breite*hoehe;
    double umfang= 2*breite+2*hoehe;
    System.out.println(„Die Fläche des Rechtecks beträgt: “ + flaeche +
    „FE und der Umfang beträgt: “ + umfang + „ LE“);
    }
}
```

Abbildung 97

```
public class FlaecheVolumenZylinderBerechnen {

public static void main(String[] args) {
    double radius=4;
    double hoehe=15;

    //was die Methode abarbeiten soll, die Variablen „radius“ und
    //„hoehe“ werden der Methode übergeben

    double volumen= Math.pow(radius)*Math.PI*hoehe;

    Die Methode gibt den Inhalt der Variablen „volumen“ zurück
    }
}
```

Abbildung 98

## Kapitel 8.5: Aufruf von Methoden (Funktionen)

Der Aufruf erfolgt gemäß der Syntax „<Funktionsname> (<Parameterliste>)“.

### Beispiel 1.

Die Methode „Test“ wird in der main-Methode aufgerufen. Der Methode wird die Variable (mit Inhalt 4) übergeben. Die Methode „test“ erwartet ein Parameter vom Typ „int“. In der Methode „test“ wird der Inhalt des übergebenen Parameters angezeigt.

```
public class Test
{
    public static void main (String args[])
    {
        int x=4;
        test(x);        //Aufruf der Methode
        System.out.println("Ausgabe in der main-Methode: x= "+x);
    }
    public static void test (int yP)
    {
        System.out.println("test-Methode: Inhalt des Parameters=
                               "+yP);
    }
}
```

Abbildung 99

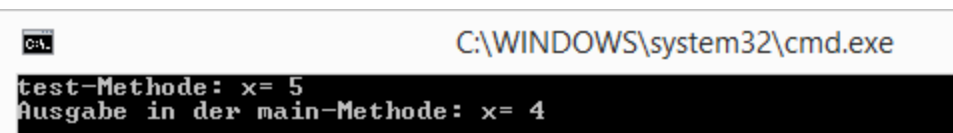


Abbildung 100

**Beispiel 2.** In einer Funktion soll für die Eingabe x den Wert  $f(x) = x*13+12$  berechnet werden.

In diesem Beispiel gibt die Funktion „funktion“ einen **int**-Wert mit Anweisung „**return wert**“ zurück und wird beendet. Der Rückgabewert der Methode wird in der Variablen „ausgabe“ gespeichert. Sollten noch Anweisungen nach einem „**return**“ stehen, so führt dies zu einer Fehlermeldung. Nach der „**return**“-Anweisung stehen keine weiteren Anweisungen!!!

```
public class funktion_2 {
    public static int funktion (int xP) {
        int wert=xP*13+12;
        return wert;
    }

    public static void main(String[] args) {
        for(int i=0; i<10; i++)
            int ausgabe = funktion(i);
        System.out.println("x= " + i + " und f(x)= " +
                               ausgabe);
    }
}
```

Abbildung 101

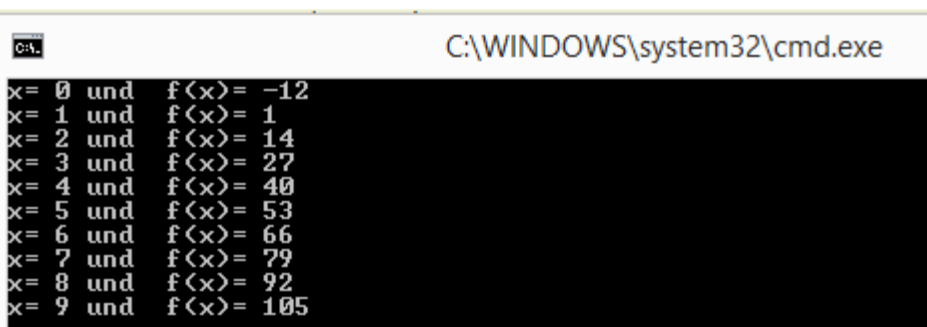


Abbildung 102

### Beispiel 3: Methode mit zwei Parameter und einen Rückgabewert

```
public class Beispiel_2{

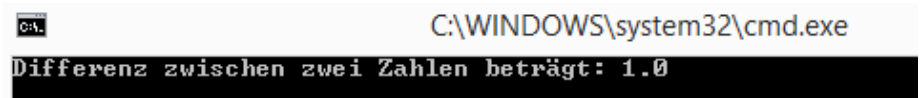
    // Da die Methode vom Datentyp double ist, wird nach Aufruf der
    //Methode eine double Zahl ausgegeben => Rückgabebetyp double
    public static double differenz(double d,double e){
        double a = d-e;
        //Anweisung return gibt den Wert der Variablen a zurück
        return a;
    }

    public static void main(String[]args){
        double zahl1=1399. 2323;
        double zahl2=1299.1413;

        //Der Rückgabewert wird in der Variablen „diffausgabe“
        //gespeichert
        double diffAusgabe = differenz(zahl1, zahl2);
        System.out.println("Differenz zwischen zwei Zahlen beträgt:
                               "+ diffAusgabe);
    }
}
```

Abbildung 103

Ergebnis:



C:\WINDOWS\system32\cmd.exe  
Differenz zwischen zwei Zahlen beträgt: 1.0

Abbildung 104



## Beispiel: Methode mit den Parameter und Rückgabewerten

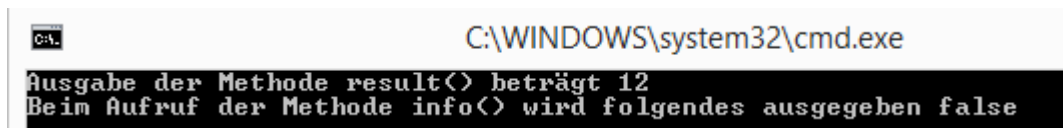
```
public class Beispiel_3{
    //Da die Methode vom Datentyp int ist, wird nach Aufruf der Methode
    // eine int Zahl ausgegeben => Rückgabebetyp int
    public static int methode(int a,int b){
        int c=5;
        int summe= a+b+c; //Verrechnen beider Werte +5

        //Anweisung return gibt den Wert der Variablen summe zurück
        return summe;
    }

    //Parameter müssen nicht den selben Rückgabebetyp wie die Methode selbst haben
    public static boolean info (int xP,int yP){
        if (xP>yP) {
            // Wichtig return um anzuweisen wo das Programm beendet wird
            //und was zurückgegeben werden soll
            return true;
        }else{
            return false;
        }
    }
    public static void main(String[]args){
        int x=3;
        int y=4;
        int result=methode(x,y);

        System.out.println("Ausgabe der Methode result() beträgt "+result);
        System.out.println("Beim Aufruf der Methode info() wird folgendes
                               ausgegeben "+info(x,y));
    }
}
```

### Ergebnis:



```
C:\WINDOWS\system32\cmd.exe
Ausgabe der Methode result() beträgt 12
Beim Aufruf der Methode info() wird folgendes ausgegeben false
```

Abbildung 105

## Kapitel 8.6: Arrays und Methoden

Natürlich kann eine Funktion auch ein Array zurückgeben. Das sieht dann so aus.

Der Ergebnistyp wird also wie eine Array-Deklaration ohne Namen geschrieben. Das Programm gibt natürlich nur eine Referenz auf das Array zurück.

Man kann auch ein Array als Parameter an eine Funktion übergeben. Als Beispiel betrachten wir Funktionen, die den Mittelwert und die Standardabweichung berechnen.

```
static double [ ] generate (int n)
// gib ein Array mit n Zufallsvariablen zurück.
{
    double R[ ]=new double[n];
    int i;
    for (i=0; i<n; i++)
    {
        R[ i ]=Math.random();
    }
    return R;
}
```

```

public class Beispiel_4{
    public static void main (String args[]){
        double R [ ]=generate(10);
        double m=mittelwert(R);
        System.out.println("Mittelwert "+m);
        System.out.println("Standard-Abweichung "+abweichung(R,m));
    }

    public static double [ ] generate (int n){
        // generiere Zufallsvektor der Länge n
        double R [ ]=new double[n];
        int i;
        for (i=0; i<n; i++){
            { R[i]=Math.random(); }
        }
        return R;
    }

    public static double mittelwert (double x[]) {
        // berechne Mittelwert von x
        int n=x.length;
        int i;
        double sum=0;
        for (i=0; i<n; i++){
            { sum+=x[ i ]; }
        }
        return sum/n;
    }

    public static double abweichung (double x[ ], double mwert){
        // berechne Standard-Abweichung von x, wenn der Mittelwert
        //bekannt ist
        int n=x.length;
        int i;
        double sum=0;
        for (i=0; i<n; i++){
            { sum+=sqr(x[i]-mwert); }
        }
        return Math.sqrt(sum/(n-1));
    }

    public static double sqr (double x){
        // Hilfsfunktion, berechne x^2
        return x*x;
    }
}

```

Abbildung 106

### Ergebnis:

Man beachte, dass nur ein einziges Array existiert. Alles, was übergeben oder zurückgegeben wird, sind nur Referenzen auf dieses Array.

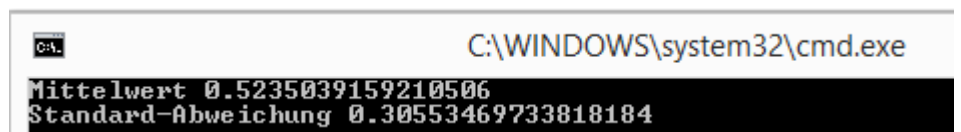


Abbildung 107

**Aufgabe 78:** Füllen Sie die folgenden Zeilen für die Methode **linie(int n)** mit Programmcode so aus, dass eine Linie mit "-" der Länge n gezeichnet wird. Setzen Sie die Methode in eine Klasse LineUtils.

```
public static void linie( int n )
{
    ...
}
```

**Aufgabe 79:** Implementieren Sie in Java folgende Funktionen (Methoden)!

- eine Methode, die den Durchschnitt zweier Zahlen zurückgibt. Der Rückgabewert wird mit *return*-Wert eingeleitet. Wähle einen passenden Datentyp aus.
- eine Methode mit Namen „fakultaet“, die Fakultät  $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$  für  $n \geq 1$  und  $0! = 1$  berechnet
- eine Methode, die das Maximum von zwei als Parameter übergebener ganzer Zahlen ermittelt  
(max(7,12) 12)

Schreiben Sie eine main-Methode, die einem Benutzer wiederholt eine Auswahl zur Ausführung der obigen Funktionen anbietet. Dabei soll das Rahmenprogramm die eingegebenen Werte auf Sinnfälligkeit überprüfen.

**Aufgabe 80:** Schreiben Sie eine Methode **quersumme**, die die Quersumme einer natürlichen Zahl *zahl* berechnet und zurückgibt. Gehen Sie davon aus, dass die eingegebene Zahl  $\geq 0$  ist. Die Quersumme einer Zahl ist die Summe ihrer Ziffern. So ergibt sich die Quersumme von *zahl*=1234 zu  $1+2+3+4=10$ .

**Tipp:** *zahl % 10 liefert dir den Teilungsrest der Division zahl/10.*

Schreiben Sie ferner eine main-Methode, die Ihre Methode mit entsprechenden Parametern aufruft.

**Aufgabe 81:** Implementieren Sie eine Java-Methode mit dem Namen „feetMeter“, die die Maßeinheit **Feet** in die Maßeinheit **Meter** umrechnet. Die Umrechnung soll nach dem Umrechnungsfaktor **1 Foot = 0.3048 Meter** erfolgen. Feet und Meter werden dabei als Gleitkommazahlen angegeben.

Schreiben Sie ferner eine neue Methode, welche eine Umrechnungstabelle von Feet nach Meter im Bereich von 1 bis 10 Feet mit Schrittweite 1 ausgibt, verwende dazu die vorher erstellte Funktion *feetMeter* und eine Schleife.

**Aufgabe 82:** Analysieren Sie folgendes Java-Programms (Abbildung 108)  
Kommentieren Sie jede Zeile.

```
import java.util.Scanner;
public class S-zahl {
    public static int reverse(int zahl) {
        int ergebnis = 0;

        while (zahl != 0) {
            ergebnis = ergebnis * 10 + zahl % 10;
            zahl = zahl / 10;
        }
        return ergebnis;
    }

    public static void main(String[] args) {
        Scanner Eingabe = new Scanner (System.in);
        System.out.println("Zahl (>=0): ");
        int eingabe = Eingabe.nextInt();
        System.out.println(reverse(eingabe));
    }
}
```

Abbildung 108

**Aufgabe 83:** Implementieren Sie ein Programm in dem vier Methoden definiert werden: addieren(), subtrahieren(), dividieren() und multiplizieren(). Alle Methoden sollen Berechnungen gemäß ihrer Namen durchführen: Zahlen addieren, subtrahieren, multiplizieren und dividieren. In den Methodenaufrufen sollen einmal zwei und einmal drei Parameter eingegeben werden. Datentyp für die Parameter: **int** und **double**. Eine Ausgabe des Ergebnisses soll zusammen mit einem kurzen Text, welche Berechnung mit welchen Zahlen vorgenommen wurde in der Main-Methode erfolgen.

**Beispiel:**

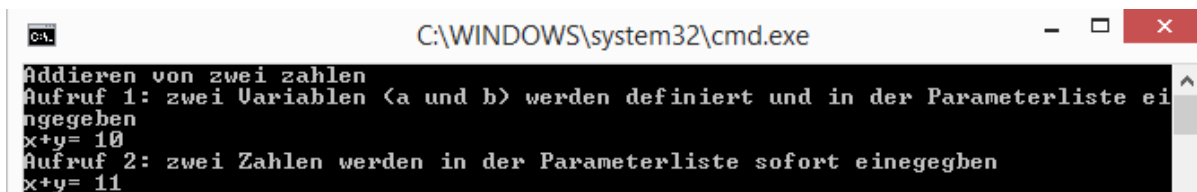
```
public class Aufgabe_Rechnen {
    public static int addieren (int x, int y) {
        int ergebnis=x+y;
        return ergebnis;
    }

    public static void main(String[] args) {
        System.out.println("Addieren von zwei zahlen");
        System.out.println("Aufruf 1: zwei Variablen (a und b) werden
                               definiert und in der Parameterliste eingegeben");

        int a= 3;
        int b=7;
        System.out.println("x+y= " + addieren(a, b));
        System.out.println("Aufruf 2: zwei Zahlen werden in der
                               Parameterliste sofort eingegeben");
        System.out.println("x+y= " + addieren(6, 5));
    }
}
```

Abbildung 109

Nach dem das Programm gestartet wurde, bekommt man folgende Ergebnisse:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.exe'. The window contains the following text:

```
Addieren von zwei zahlen
Aufruf 1: zwei Variablen (a und b) werden definiert und in der Parameterliste ei
ngegeben
x+y= 10
Aufruf 2: zwei Zahlen werden in der Parameterliste sofort eingegeben
x+y= 11
```

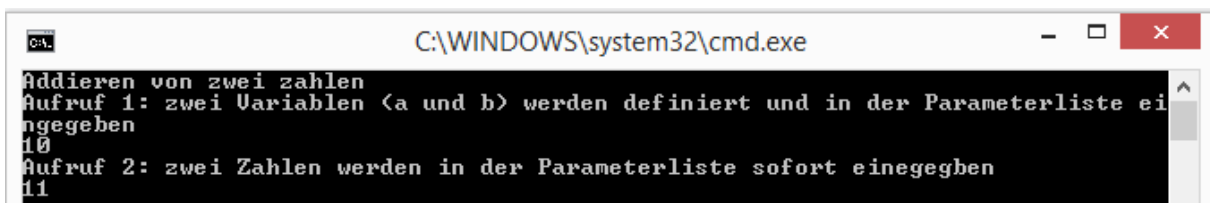
Abbildung 110

**Aufgabe 84:** Implementieren Sie ein Programm in dem vier Methoden definiert werden: addieren(), subtrahieren(), dividieren() und multiplizieren(). Also wie in der vorherigen Aufgabe, aber alle Methoden liefern kein Wert zurück. Alle Methoden sollen Berechnungen gemäß ihrer Namen durchführen: Zahlen addieren, subtrahieren, multiplizieren und dividieren. In den Methodenaufrufen sollen einmal zwei eingegeben werden. Datentyp für die Parameter: **int** und **double**. Eine Ausgabe des Ergebnisses soll zusammen mit einem kurzen Text, welche Berechnung mit welchen Zahlen vorgenommen wurde in der jeweiligen Methode erfolgen. Zum Beispiel hier die addieren-Methode als void-Methode:

```
public class Aufgabe_Rechnen_void {  
    public static void addieren (int x, int y) {  
        int ergebnis=x+y;  
        System.out.println(ergebnis);  
    }  
  
    public static void main(String[] args) {  
        System.out.println("Addieren von zwei zahlen");  
        System.out.println("Aufruf 1: zwei Variablen (a und b) werden  
            definiert und in der Parameterliste eingegeben");  
  
        int a= 3;  
        int b=7;  
        addieren(a, b);  
        System.out.println("Aufruf 2: zwei Zahlen werden in der  
            Parameterliste sofort eingegeben");  
        addieren(6, 5);  
    }  
}
```

Abbildung 111

Nach dem das Programm gestartet wurde, bekommt man folgende Ergebnisse:



```
C:\WINDOWS\system32\cmd.exe  
Addieren von zwei zahlen  
Aufruf 1: zwei Variablen (a und b) werden definiert und in der Parameterliste ei  
ngegeben  
10  
Aufruf 2: zwei Zahlen werden in der Parameterliste sofort eingegeben  
11
```

**Aufgabe 85:** Überlegen Sie, was das folgende Programm macht. Was erwarten Sie als Ausgabe? Führen Sie das Programm aus und vergleichen Sie das Ergebnis mit Ihren Erwartungen.

```
public class Aufgabe1 {
    public static void main(String[] arguments) {
        print(1, "Martin");
        print(2, "Arthur");
        print(3, "Florian");
        print(4, "Robert");
    }

    public static void print(int zahl, String name) {
        System.out.println(zahl + ": " + name);
    }
}
```

Abbildung 112

**Aufgabe 86:** Überlegen Sie, was drei die folgende Programme machen. Was erwarten Sie als Ausgabe? Führen Sie die Programme aus und vergleichen Sie das Ergebnis mit Ihren Erwartungen.

a)	b)
<pre>public class <b>Aufgabe_2_a</b> {     public static void main(String[] arguments) {         System.out.println(gibMirEineZahl());     }      public static double gibMirEineZahl() {         return 15.3;     } }</pre>	<pre>public class <b>Aufgabe_2_b</b> {     public static void main(String[] arguments) {         System.out.println("2 + 7 * 3 = " +             add(2, 7, 3));         System.out.println("8 + 0 * 2 = " +             add(8, 0, 2));         System.out.println("-7 + 7 * 4 = " +             add(-7, 7, 4));     }      public static int add(int int1, int int2, int int3) {         return int1 + int2 * int3;     } }</pre>

c)
<pre>public class <b>Aufgabe_2_c</b> {     public static void main(String[] arguments) {         System.out.println(mathematik(1, 2));         System.out.println(mathematik(1, 5));         System.out.println(mathematik(3, 4));     }     public static boolean mathematik(int argument1, int argument2) {         return (argument1 + 5) &lt; (argument2 * 2);     } }</pre>

**Aufgabe 87:** Schreiben Sie eine Methode, welche entscheidet, ob es sich bei der übergebenen Jahreszahl um ein Schaltjahr handelt, oder nicht. Das Jahr könnte man als Integerzahl (**int**) übergeben, der Rückgabewert der Methode sollte ein Wahrheitswert (**boolean**) sein.

Die Methode könnte so aussehen:

```
public static boolean istSchaltjahr(int jahr) {           ... }
```

Die Eigenschaften eines Schaltjahres sind

- Ein Jahr ist kein Schaltjahr, wenn die Jahreszahl nicht durch 4 teilbar ist.
- Ein Jahr ist ein Schaltjahr, wenn die Jahreszahl durch 4, aber nicht durch 100 teilbar ist.
- Ein Jahr ist ebenfalls ein Schaltjahr, wenn die Jahreszahl durch 4, durch 100 und durch 400 teilbar ist.

Das Programm soll mit einer Benutzerfreundlichen Interaktion ausgestattet sein. Dazu müssen Sie die Klasse **Scanner** verwenden. Ziel der Aufgabe ist es, dass der Benutzer auf der Konsole aufgefordert wird, eine Jahreszahl einzugeben und daraufhin eine Antwort bekommt, ob die eingegebene Jahreszahl ein Schaltjahr ist, oder nicht.

## Kapitel 8.7: Der Gültigkeitsbereich von Variablen

Wie andere Programmiersprachen bietet auch Java die Möglichkeit, Variablen global für das ganze Programm oder auch nur lokal in einzelnen Funktionen zu definieren.

Wenn man eine Variable, wie es gewohnt ist, außerhalb einer Methode deklariert, ist diese im gesamten Programm bekannt, d.h. der Interpreter bezeichnet mit diesem Variablennamen immer den gleichen Speicherplatz, und wenn man etwas hineingeschrieben hat, kann man diesen Wert im gesamten Programm wieder auslesen.

Wenn man nun eine Variable innerhalb einer Methode, also zwischen der öffnenden und der schließenden geschweiften Klammer, deklariert, ist die Variable nur innerhalb der Funktion bekannt. Sie wird dann als lokale Variable bezeichnet.

**Beispiel 1:** In folgendem Programmstück wird eine globale String-Variable „**trennsymbol**“ deklariert und mit dem Wert " / " belegt. Die nachfolgende Methode **verkette()** definiert den Parameter „s“ und den Parameter „t“. Die Methode **verkette()** fügt zwischen zwei Wörtern das Trennsymbols „/“.

Ein Aufruf der Methode „**verkette**(„Hallo“, „Welt“)“ liefert folgendes Ergebnis „Hallo / Welt“. Das Ergebnis wird mit der Hilfe der Anweisung „System.out.println()“ ausgegeben.

Ein Aufruf der Methode, etwa mit **s=verkette**(„alpha“, „beta“) liefert als Methodenwert das Ergebnis „alpha / beta“. Dieser wird hier der Variablen **s** zugewiesen.

Ausgabe des Programms:

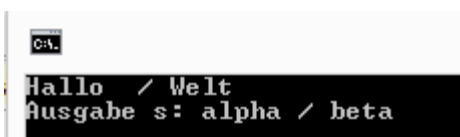


Abbildung 114

```
public class Beispiel_1
{
    static String trennsymbol=" / "; //Globale Variable

    public static void main(String[] args) {

        System.out.println(verkette("Hallo ", "Welt"));
        String s=verkette("alpha", "beta");
        System.out.println("Ausgabe s: "+s);
    }

    static String verkette(String s, String t)
    {
        return s+trennsymbol+t;
    }
}
```

Abbildung 113



Also das Programm tut das, was man erwarten würde. Normalerweise haben innerhalb einer Methode sowohl die dort deklarierten lokalen Variablen als auch die außerhalb der Methode deklarierten globalen Variablen Gültigkeit. Ein Problem tritt auf, wenn eine globale Variable und eine lokale Variable denselben Namen haben. Wenn also z.B. beide Variablen *x* heißen und wenn dann innerhalb der Funktion auf *x* zugegriffen wird, dann ist zunächst unklar, welches *x* gemeint ist. **Daher gilt die Regel, dass innerhalb der Funktion die lokale Variable Vorrang hat.** Die Konsequenz ist, dass innerhalb der Funktion die globale Variable gleichen Namens nicht erreichbar ist.

**Beispiel 2.** Eine Version der Methode *verkette*.

Es werden zwei Variablen deklariert, eine **globale** Variable *trennsymbol* und eine **lokale** Variable mit demselben Namen.

Die **globale** Variable *trennsymbol* erhält den Wert " / ", die **lokale** Variable *trennsymbol* erhält den Wert "+".

Innerhalb der Methode hat die **lokale** Variable *trennsymbol* Vorrang.

Immer, wenn innerhalb der Methode auf die Variable *trennsymbol* zugegriffen wird, ist die lokale Variable *trennsymbol* gemeint, und diese hat den Wert " + ".

Ausgabe des Programms:

*verkette*("alpha", "beta");, wird das Ergebnis "alpha+beta" zurückgegeben.

```
public class Beispiel_2
{
    static String trennsymbol=" / ";    //Globale Variable
    public static void main(String[] args) {
        System.out.println(verkette("Hallo ", "Welt"));
        String s=verkette("alpha", "beta");
        System.out.println("Ausgabe s: "+s);
    }
    static String verkette(String s, String t)
    {
        String trennsymbol=" + ";    //lokale Variable
        return s+trennsymbol+t;
    }
}
```

Abbildung 115

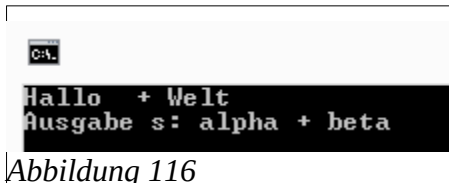

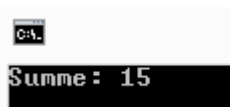
A screenshot of a Java program's output. It shows two lines of text: "Hallo + Welt" and "Ausgabe s: alpha + beta". The text is displayed in a monospaced font on a dark background, typical of a terminal window.

Abbildung 116

### Beispiel 3. Lokale und Globale Variable

In der Klasse „OhneGlobaleVar“ wird die Variable „sum“ in jeder Methode neu definiert. In der Klasse „MitGlobalerVar“ wird die Variable „sum“ nur ein mal definiert und zwar nur als globale Variable.

Das Programm ohne globale Variable	Das Programm mit einer globalen Variable
<pre> <b>public class</b> OhneGlobaleVar {  //Keine Globale Variable     <b>public static</b> void add(int x) {         <b>int</b> sum = 0; //lokale Variable         sum = sum + x;     }      <b>public static</b> void main(String[] args) {         <b>int</b> sum = 0; //lokale Variable         add(3);         add(5);         add(7);         System.out.println("Summe:                                "+sum);     } }         </pre> <p>Abbildung 117</p>	<pre> <b>public class</b> MitGlobalerVar {     <b>public static</b> <b>int</b> sum = 0; //Globale Variable      <b>public static</b> void add(int x) {         sum = sum + x; //lokale Variable     }      <b>public static</b> void main(String[] args) {         add(3);         add(5);         add(7);         System.out.println("Summe: "+sum);     } }         </pre> <p>Abbildung 118</p>
Ausgabe des Programm:	Ausgabe des Programm:
 <p>Abbildung 119</p>	 <p>Abbildung 120</p>

**Beispiel 4.** Wir betrachten die folgende Methode, die ein Mittelwert berechnet, mit einem **double-Feld** als **Parameter** und einer **double-Zahl** als **Rückgabewert**.

**i, n, w** und auch der Parameter **x** sind **lokale Variablen** dieser Methode. Für sie wird erst Platz geschaffen, wenn die Methode aufgerufen wird.

Vorher und nachher sind diese Variablen bzw. ihre Namen unbekannt.

**Globale** Variablen der Applikationsklasse **Mittelwert** (diese müssen als **static** deklariert werden), die diese Methode enthält, können ohne weiteres dieselben Bezeichner als die lokalen Variablen haben (auch wenn dies nicht immer sinnvoll ist):

```

public class Mittelwert {
    static double mittelwert(double[] x){
        int i, n;      //lokale Variable
        n=x.length;  //lokale Variable
        double w=0;  //lokale Variable

        for (i=0;i<n;i++)
        {
            int zahl_aus_dem_array=x[i];
            w+=zahl_aus_dem_array;
        }
        w/=n;
        return w;
    }
}
        
```

Abbildung 121

Hier werden unnötiger Weise eine globale und eine lokale Variable gleichen Namens „w“ verwendet, die unterschiedliche Werte haben.

Ausgegeben wird im main()-Block die globale Variable.

Die lokalen Variablen der Methode mittelwert() sind im main()-Block unbekannt.

Wollte man auf sie (z.B. auf n) im main()-Block zugreifen, so gäbe es eine Syntax-Fehlermeldung „Undefined variable: n“.

```
class Mittelwert{
    static double v, w;    //globale Variablen in der Klasse Mittelwert

    static double mittelwert(double[] x){
        int i, n;
        n=x.length;
        double w=0;

        for (i=0;i<n;i++) {w+=x[i];}

        w/=n;
        return w;        //w=berechneter Mittelwert
    }

    public static void main(String[] args){
        double[] y={1, 2.1, 3};
        w=y[0];           //hat nichts mit dem Mittelwert zu tun
        v=mittelwert(y);   //Hier wird der Mittelwert mit v bezeichnet
        System.out.println("w="+w+" Mittelwert="+v); // Ausgabe: w, v
    }
}
```

Abbildung 122

Merke: Globale Variable können in einem Methodenblock verändert werden, wenn es keine lokalen Variablen gleichen Namens gibt.

Ausgabe:

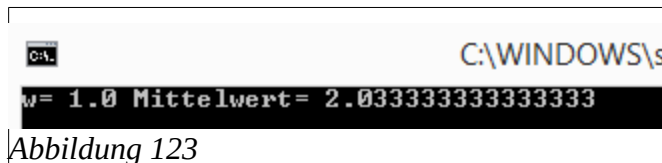


Abbildung 123

**Beispiel 5.** Das Programm erfüllt denselben Zweck wie das Programm aus dem Beispiel 3 ist aber kürzer.

Dieses Programm ist sogar kürzer, aber weniger gut strukturiert.

Der Mittelwert wird jetzt implizit berechnet.

Die Methode **mittelwert()** hat **keinen Rückgabewert (eine void-Methode)**.

Übrigens die **lokale Feldvariable double[] y** der main()-Methode könnte den gleichen Namen wie die **Parametervariable double[] x** der **mittelwert()-Methode** haben (Speicherplatz wird dadurch nicht gespart).

```
Zeile 1: class Mittelwert2{
Zeile 2:     static double w;        //globale Variable
Zeile 3:     static void mittelwert(double[] x){
Zeile 4:         int i, n;          //lokale Variablen
Zeile 5:         n=x.length;       //lokale Variablen
Zeile 6:         for (i=0;i<n;i++) { w+=x[i]; }
Zeile 7:         w/=n;
Zeile 8:     }

Zeile 9:     public static void main(String[] args){
Zeile 10:         double[] y={1, 2.1, 3};
Zeile 11:         mittelwert(y);
Zeile 12:         System.out.println("Mittelwert="+w); //Ausgabe
Zeile 13:     }
Zeile 14: }
```

Kommentierung jeder Zeile:

In der Zeile 1 steht der Klassenkopf. Die Klasse trägt den Namen „Mittelwert2“. In der Zeile 2 wird eine globale Variable mit dem Namen „w“ vom Typ „double“ definiert und ist mit dem Modifikator „static“ gekennzeichnet, d.H. diese Variable ist eine Klassenvariable (Zur Info: Sie sind nicht den von der Klasse abgeleiteten Objekten zugeordnet, sondern gehören zur Klasse selbst und stehen in allen Instanzen gleichermaßen zur Verfügung.). In der Zeile 3 wird die Methode „mittelwert“ mit dem Parameter „x“, ein eindimensionales Array, vom Typ „double“ definiert. In der Methode werden die lokalen Variablen i und n vom Typ „integer“ definiert. In der Zeile 4 wird der Variablen n die Länge des Arrays „x“ zugewiesen. Dann beginnt in der Zeile 5 ein for-Schleife mit der Laufvariablen „i“, die bei „0“ beginnt und nach jedem Durchlauf um eins hochgezählt wird. Die Schleife läuft so lange so lange i kleiner als n ist. Im Schleifenkörper (oder Schleifenrumpf; in den Geschweiften Klammern der for-Schleife) wird der Variablen „w“ nach jedem Durchlauf die nächste Zahl im Array „x“ zugewiesen. Nach Beendigung der for-Schleife wird die Zahl in der Variablen „w“ durch die Zahl in der Variablen „n“ dividiert und dieses Ergebnis wird der Variablen „w“ zugeordnet (also „w“ wird überschrieben).

Ausgabe des Programm:

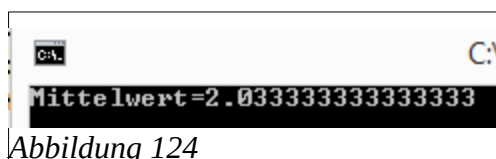


Abbildung 124

**Aufgabe 88:** Erklären Sie, was zu bedenken ist, wenn eine Variable außerhalb der Methoden deklariert wird.

**Aufgabe 89:** Erklären Sie, was zu bedenken ist, wenn eine Variable innerhalb von Methoden deklariert.

**Aufgabe 90:** Erklären Sie, was man unter einer lokalen Variablen versteht.

**Aufgabe 91:** Erklären Sie, was man unter einer globalen Variablen versteht.

**Aufgabe 92:** Eine lokale Variable und eine globale Variable haben die selben Bezeichnungen. Nehmen Sie Stellung zu dieser Aussage.

**Aufgabe 93:** Erklären Sie die Bedeutung der Operatoren „/=“ und += im Programm in der Abbildung 121.

**Aufgabe 94:** Erstellen Sie für die for-Schleife im Programm in der Abbildung 121 eine Belegungstabelle. Gehen Sie davon aus, dass im Array die folgenden Zahlen enthalten sind: {4, 5, 1, 2, 6, 2, 2}

	i	n	zahl_aus_dem_array	w
Nach der Initialisierung				
1. Schleifendurchlauf mit i =				
2. Schleifendurchlauf mit i =				
3. Schleifendurchlauf mit i =				
4. Schleifendurchlauf mit i =				
5. Schleifendurchlauf mit i =				
6. Schleifendurchlauf mit i =				
7. Schleifendurchlauf mit i =				

Abbildung 125: Belegungstabelle

**Aufgabe 95:** Erklären Sie begründet, was bei der Definition von lokalen Variablen zu beachten ist. Implementieren Sie ein Beispiel dazu und fügen Sie dem Programm Kommentarzeilen hinzu, die auf die Problematik hinweisen.

**Aufgabe 96:** Erklären Sie, warum das Programm in der Abbildung 119 die Ausgabe „Summe=0“ liefert, aber das Programm in der Abbildung 120 die Ausgabe „Summe = 15“ liefert.

**Aufgabe 97:** Das folgende Programm (Abbildung 126) liefert beim Ausführen eine Fehlermeldung.

- Erklären Sie begründet, warum das Programm eine Fehlermeldung ausgibt.
- Kommentieren Sie das Programm. Benutzen Sie beim Kommentieren entsprechende Fachwörter.
- Wie könnte man es verändern, so dass das Programm korrekt arbeitet?

**Aufgabe 98:** Analysieren Sie folgendes Java-Programm (Abbildung 127) und geben Sie an, was das Programm ausgibt.

**Aufgabe 99:** Analysieren Sie folgendes Java-Programm (Abbildung 128), kommentieren Sie jede Zeile (Was macht die entsprechende Zeile) und geben Sie an, was das Programm ausgibt und warum. Ersetzen Sie im Programm die Zeile „arr[i] += 10;“ durch die Zeile: „arr[i] \*= 10;“. Wie hat sich die Ausgabe vom Programm geändert?

```
public class LokaleVariable {
    static void bestimmeLebensdauer(){
        int lebensDauer;
        System.out.println(lebensDauer);
    }
    public static void main (String [] args) {
        bestimmeLebensdauer();
    }
}
```

Abbildung 126

```
public class Aufgabe_2{
    public static void main(String[] args){
        int[] testArr=new int[2];
        testArr[0]=1;
        testArr[1]=2;
        System.out.println(testArr[0] + " " + testArr[1]);
        swap(testArr);
        System.out.println(testArr[0] + " " + testArr[1]);
    }

    public static void swap(int testArr[ ]){
        int c = testArr[0];
        testArr[0]=testArr[1];
        testArr[1]=c;
    }
}
```

Abbildung 127

```
public class Aufgabe_3    {
    public static void addTen(int[] arr)
    {
        for(int i = 0; i < arr.length; i++)    {
            arr[i] += 10;
        }
    }

    public static void main(String[] args) {
        int[] arr = {1, 3, 5, 7, 9};
        addTen(arr);
        for(int i = 0; i < arr.length; i++)
        {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Abbildung 128

**Aufgabe 100:** Analysieren Sie folgendes Java-Programm, kommentieren Sie jede Zeile (Was macht die entsprechende Zeile) und geben Sie an, was das Programm ausgibt und warum.

Erstellen Sie zusätzlich eine Methode **produkt()**, in der das Produkt aller Elemente des Arrays berechnet wird.

**Aufgabe 101:** Analysieren Sie folgendes Java-Programm, kommentieren Sie jede Zeile (Was macht die

entsprechende Zeile) und geben Sie an, was das Programm ausgibt und warum. Wo liegt der Unterschied zwischen dem Programm in der Abbildung 130 und dem Programm in der Abbildung 129.

```
public class Aufgabe_4    {
    public static void sum(int[] arr)    {
        int sum = 0;
        for(int i = 0; i < arr.length; i++)
        {
            sum = sum + arr[i];
        }
        System.out.print("Summe von Arrayelementen ist : " + sum);
    }
    public static void main(String[] args) {
        int[] arr = { 1, 2, 3, 4, 5};
        sum(arr);
    }
}
```

Abbildung 129

```
public class Aufgabe_5{
    public static void sum(int[][] arr)    {
        int sum = 0;
        for(int i = 0; i < arr.length; i++)
        {
            for(int j = 0; j < arr[0].length; j++)
            {
                sum = sum + arr[i][j];
            }
        }
        System.out.print("Summe von Arrayelementen ist : " + sum);
    }
    public static void main(String[] args)    {
        int[][] arr = {
            {1, 2, 3, 4, 5},
            {2, 4, 6, 8, 10},
            {1, 3, 5, 7, 9}
        };
        sum(arr);
    }
}
```

Abbildung 130

**Aufgabe 102: Notenverwaltung**

Schreiben Sie ein Programm zur Verwaltung der Noten einer Schulklasse. Erstellen Sie drei Methoden. Eine Methode mit dem Namen „einLesen“, eine weitere mit dem Namen „ausgebenListe“ und eine Methode mit dem Namen „durchSchnitt“.

Name der Methode	Rückgabetype	Parameterliste	Beschreibung
einLesen	double[]	Anzahl SchülerInnen: int anzahl	Noten der Klausur einlesen lassen, wobei die Klassengröße interaktiv in Erfahrung gebracht wird. Die Notenliste sollte über ein Zufallszahlengenerator gefüllt werden (spart Zeit).
ausgebenListe	void	double[]	Notenliste der Klausur ausgeben lassen
durchSchnitt	double	double[]	Durchschnittsnote der Klasse ermitteln lassen.

## Kapitel 9: Klassen und Objekte, Instanzieren von Objekten einer Klasse

Bis her haben wir das Programm in nur einer Klasse implementiert. Jetzt verwenden wir mehrere Klassen um ein Programm zu implementieren. In der Abbildung 135 und Abbildung 136 ist ein Beispiel für die Verwendung von zwei Klassen.



Abbildung 131  
Platon



Abbildung 134



Abbildung 133

Bauplan Hund → Klasse

Eigenschaften: → Variablen

+ Vier Beine

+ Fell

Verhalten: → Methoden

+ Laufen

+ Bellen

+ Stock holen

Abbildung 132:

Von Platon (427 - 347 v. Chr.), dem griechischen Philosophen der Antike stammt sinngemäß folgender Satz: Alle konkreten Gegenstände und Sachverhalte, die der Mensch mit seinen Sinnen wahrnimmt, sind pure (Ab)bilder jener Urbilder und werden nur erkannt, indem sie als eben solche (Ab)bilder identifiziert werden. oder: Ein "java-sprechender" Platonschüler Alles was der Mensch wahrnimmt sind Objekte, also Instanzen von Klassen (= Ideen). Was ist eine Klasse, was ist ein Objekt Wenn ein Kind sagt, dass der vor ihm stehende Dackel "Fifi" ein Hund ist, dass aber auch der Schäfer "Fakir" ein Hund ist, dann hat das Kind gelernt, zwischen Klassen und Objekten im Sinne von OOP (= Objekt-Orientierte-Programmierung) zu unterscheiden. Den "Hund" gibt es tatsächlich nicht, der ist lediglich eine Abstraktion eine Idee. Wirklich existieren nur "Fifi" und "Fakir". In der OOP-Sprache sagen wir auch "Fifi" und "Fakir" sind Instanzen der Klasse 'Hund'. Die Rolle von Klasse und Objekt und ihre Beziehung untereinander können wir uns noch durch ein weiteres Bild veranschaulichen. So sagt der Informatiker: **Eine Klasse ist wie ein Bauplan, das Objekt ist dann eine konkrete Umsetzung.**

Was gehört nun alles zu einem solchen Bauplan? Er beschreibt zuerst einmal alle Eigenschaften (Attribute, Klassenvariablen), die ein Objekt haben soll, das man mit Hilfe dieses Bauplans erzeugt. Weiter enthält der Bauplan, also die Klasse die Methoden (steuern das Verhalten des Objekts), mit denen man Veränderungen an den konkreten Objekten vornehmen kann.

Kommen wir nun zu einem konkreten Beispiel. Ein Tierheim möchte, die sich im Tierheim befindlichen Tiere verwalten. Wir fangen damit an, die Hunde in eine Klasse abzubilden. Es werden zwei Klassen benötigt. In der einen Klasse (sie wird mit „Tiere“ benannt) befindet sich die main-Methode. Ohne die main-Methode kann kein Programm starten. Die andere Klasse wird mit Hunde benannt. In der Abbildung 135 sehen Sie die Klasse „Tiere“ und in der Abbildung 136 sehen Sie die Klasse „Hunde“.

Die Klasse „Tiere“ hat die Funktion (die Aufgabe) aus dem Bauplan „Hunde“ konkrete einzelne „Hunde“ zu erzeugen (zu instanzieren). Das geschieht in der Zeile 7 in der Abbildung 135. Die konkrete Umsetzung (Instanziierung) eines Bauplans (Klasse) zu einem Objekt wird durch den Operator „new“ realisiert:

```
Hunde fifi = new Hunde(); //Das Objekt heisst hier „fifi“ und ist vom Typ „Hunde“
```

Mit dem Punktoperator (nach dem Objektnamen kommt der Punkt) wird die Methode eines Objektes aufgerufen (Zeile 10 bis 13 in der Abbildung 136):

```
fifi.laufen(„15 km“);  
fifi.setFellfarbe („grau“);
```



fifi.setFellfarbe ();

```
1 public class Tiere{
2     //AnfangAttribute
3     //EndeAttribute
4
5     public static void main(String[] args){
6         //Objekt instanziiieren
7         Hunde fifi = new Hunde();
8
9         //Festlegung der Eigenschaften
10        fifi.fellfarbe="grau";
11        fifi.name="fifi-zwo";
12        fifi.alter="2";
13        fifi.laufen();
14    }
15 }//end of Tiere
16
```

Abbildung 135

Nur in der Klasse Tiere (die Hauptklasse) befindet sich die main-Methode

Instanziierung eines Objekts der Klasse Hunde.

Festlegung der Attribute, Festlegung der Eigenschaften des Objekts.

Aufruf der Methode „laufen“ von der Klasse Hunde.

```

public class Hunde {
    //AnfangAttribute

    public String fellfarbe;
    public String name;
    public int alter;

    //EndeAttribute

    public Hunde(){
        name="kittiblau"
        fellfarbe=blau;
        alter=5; //in jahren
    }

    public String getFellfarbe(){
        return fellfarbe;
    }

    public void setFellfarbe(String fellfarbeP){
        fellfarbe=fellfarbeP;
    }

    public void laufen(String streckeP){
        System.out.println("Der Hund mit dem Namen: „+ this.name + “läuft die Strecke: “ +
        streckeP);
    }

    public void gibInfoHund(){
        System.out.println("Der Hund mit dem Namen: „+ this.name +
        “ hat die Fellfarbe: “ + this.fellfarbe);
    }

}

```

Der Standardkonstruktor.

Festsetzung von Standarteigenschaften für jedes Objekt.

Die Methode „laufen“, die das Verhalten des Objekts steuert.

Die Methode „gibInfoHund“, die alle Daten des Objekts anzeigt.

//end of Hunde

Abbildung 136

**Aufgabe 103:** Instanziiieren Sie vier weitere Objekte der Klasse Hunde. Die Objektnamen lauten „beloHund“, „kikiHund“, „martinchenHund“ und „kirchyHund“. Fellfarbe, Name und Alter können Sie selber bestimmen.

Die Hunde sollen auch natürlich unterschiedliche Strecken laufen.

**Aufgabe 104:** Erweitern Sie die Klasse Hunde um ein weitere Attribute mit Namen „futter“ und „besitzer“. Beide vom Typ „String“. Nicht vergessen auch den Standardkonstruktor zu ergänzen. Ergänzen Sie auch die Methode „gibInfoHund“, damit alle Informationen angezeigt werden.

**Aufgabe 105:** Speichern Sie alle Dateien zur Anwendung „Tiere“ in ein neues Verzeichnis mit Namen „TiereV1“. Erstellen Sie eine neue Klasse mit Namen „Katze“. Implementieren Sie alle Eigenschaften, die eine Katze so haben kann, wie „fellfarbe“, „gewicht“, „rasse“, „geburtsdatum“, usw... Implementieren Sie typische Verhaltensweisen wie „trinken(String fluessigkeitP, int mengeP)“, „fressen(String futterP, int mengeP)“, „klettern(String objektP, int hoeheP)“, usw... einer Katze. Erzeugen Sie drei verschiedene Katzen. Geben Sie alle Informationen zu den Katzen auf die Konsole aus.

**Aufgabe 106:** Speichern Sie alle Dateien zur Anwendung „Tiere“ in ein neues Verzeichnis mit Namen „TiereV2“. Speichern Sie die erzeugten Tiere jeweils in einem Array, also die Katzen in einem „katzenArray“ und die Hunde in einem „hundeArray“. Alle Informationen über die Tiere sollen weiterhin auf der Konsole angezeigt werden.

**Aufgabe 107:** In der folgenden Tabelle sollen Fachbegriffe, die Sie kennen müssen, erklärt werden.

Fachbegriff	Erklärung
Konstruktor	
Variable	
Methode	
Parameter	
New Operator	
Instanziierung	
Bauplan	
Typ	
Attribute	
Punktooperator	
Fallunterscheidung	
forschleife	
whileschleife	
String	
int	
Objektvariable	
Klassenvariable	

**Aufgabe 108:** Erstellen Sie ein Verzeichnis mit dem Namen „geometrische\_figuren“. In diesen Verzeichnis sollen alle Java Dateien abgespeichert werden. Es soll eine Anwendung zu geometrischen Formen entwickelt werden. In der Abbildung 137 und Abbildung 138 sehen Sie die dazugehörigen Klassenkarten. Wie man Klassenkarten „liest“ (Mit einer Klassenkarte wird der Aufbau und Eigenschaften einer Klasse visuell dargestellt) zeigt das Kapitel 9.2:.

- a) Implementieren Sie die Klassen „Kreis“, „Rechteck“ und „Punkt“. In der Methode „flaecheninhalt“ soll die entsprechende Fläche ausgerechnet und das Ergebnis dem Aufrufer zurückgeben werden. In der Methode „umfang“ soll der entsprechende Umfang ausgerechnet und das Ergebnis dem Aufrufer zurückgeben werden.

In der Methode „gibinformationenAus“ werden alle Informationen zu der geometrischen Form ausgegeben werden.

Kreis
<ul style="list-style-type: none"> <li>■ radius: double</li> <li>■ <b>mittelpunkt: Punkt</b></li> </ul>
<ul style="list-style-type: none"> <li>⊙ Kreis(pXKoord: double, pYKoord: double, pRadius: double)</li> <li>⊕ flaecheninhalt(): double</li> <li>⊕ umfang(): double</li> <li>⊕ gibinformationenAus(): String</li> <li>⊕ getMittelpunkt(): Punkt</li> </ul>

Abbildung 137

b) Die Klasse in der die main-Methode enthalten ist soll „FigurTest“ heißen.

c) In der main-Methode sollen fünf unterschiedliche Rechtecke und fünf unterschiedliche Kreise erstellt werden. Alle Kreise werden in einem Array abgespeichert. Die Rechtecke werden auch in ein eigenes Array abgespeichert.

Rechteck
<div> <div></div> <div>breite: double</div> </div> <div> <div></div> <div>hoehe: double</div> </div>
<div> <div></div> <div>Rechteck(pXKoord: double, pYKoord: double, pBreite: double, pHoehe: double)</div> </div> <div> <div></div> <div>flaecheninhalt(): double</div> </div> <div> <div></div> <div>umfang(): double</div> </div> <div> <div></div> <div>gibInformationenAus(): String</div> </div>

Abbildung 138

d) Mit Hilfe einer for-Schleife sollen alle Informationen zu allen Rechtecke und Kreise ausgegeben werden.

e) Mit Hilfe einer while-Schleife sollen alle Informationen zu allen Rechtecke und Kreise ausgegeben werden.

Punkt
<div> <div></div> <div>Punkt(pXKoord: double, pYKoord: double)</div> </div> <div> <div></div> <div>flaecheninhalt(): double</div> </div> <div> <div></div> <div>umfang(): double</div> </div>

Abbildung 139

f) Implementiert eine weitere Figur: Dreieck. Auch zu dieser Figur sollen Umfang und Fläche berechnet werden. Erstellen Sie auch für diese Figur 5 entsprechende Objekte, die in ein eigenes Array abgespeichert werden.

**Aufgabe 109:** Erstellt ein Verzeichnis mit dem Namen „geometrische\_figurenV1“. In diesen Verzeichnis sollen alle Java Dateien der Anwendung „geometrische\_figuren“ abgespeichert werden. Erweitern Sie die Anwendung um eine Weitere Klasse mit dem Namen „ZusammengesetzteFigur“. Die Klasse enthält drei Arrays (Objektvariablen): eins für die Dreiecke, eins für die Rechtecke und eins für die Kreise. Mit den Methoden „setDreieck(Dreieck dreieckP)“, „setRechteck(Rechteck rechteckP)“ und „setKreis(Kreis kreisP)“ werden die einzelnen Bestandteile der zusammengesetzten Figur gespeichert. Diese drei Methoden sollen automatisch feststellen an welcher Stelle im Array die übergebene Figur eingetragen wird. Die Methode „gibInformationenAus(String txtP)“ gibt aus aus wie vielen einzelnen geometrischen Figuren die zusammengesetzte Fläche besteht, wie groß die Gesamtfläche ist und gibt den Inhalt der Variablen „txtP“ aus.

## Kapitel 9.1: Zugriffsmodifikatoren (Sichtbarkeit)

Allgemein erklärt gibt es 4 Arten von Zugriffsmodifikatoren:

### 1. **Private**

Wenn eine Variable oder Methode **private** deklariert ist, dann ist sie nur in der Klasse in der sie erstellt wurde sichtbar, also nicht von anderen Klassen.

### 2. **Public**

Wenn eine Variable oder Methode **public** deklariert ist, dann ist sie von jeder Klasse aus sichtbar und bearbeitbar, die eine Referenz(ein Objekt) zu der Klasse der Variable/Methode besitzt.

### 3. **Default**

Wenn eine Variable oder Methode **default** deklariert ist, also keinen Zugriffsmodifikator hat, dann wird sie vom Paket in dem sie befindlich ist gesehen und kann via Referenz auf das Objekt der Klasse verändert/gesehen werden.

### 4. **Protected**

Wenn eine Variable oder Methode **protected** deklariert ist, dann kann nur vom gleichen Paket, oder von Klassen die von der Klasse der Variable/Methode erben, aus darauf zugegriffen werden. Was erben bedeutet, bzw. Vererbung ist, erkläre ich in einem späteren Tutorial.

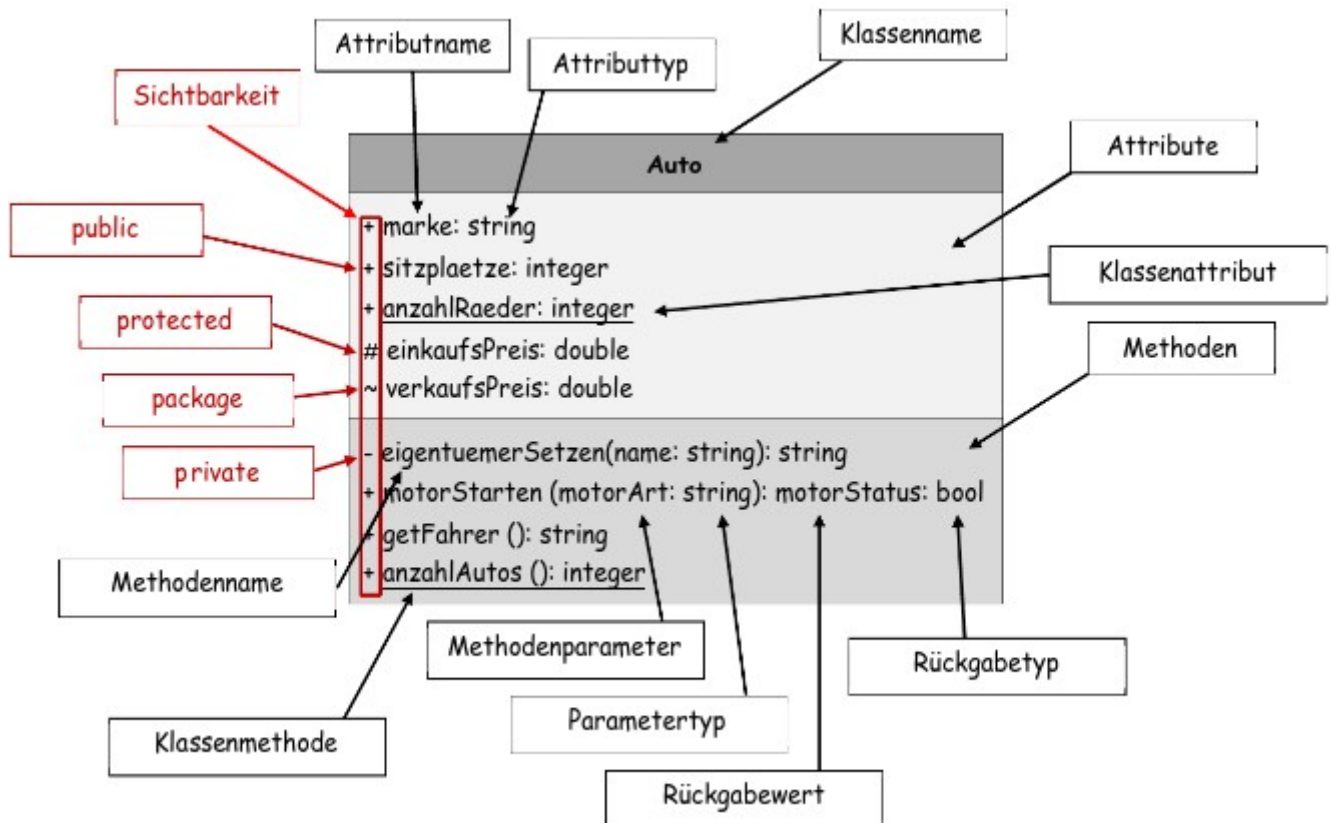
Bisher sind alle Eigenschaften in den Klassen als „**public**“ gekennzeichnet. Das heißt, jede andere Klasse kann auf diese Eigenschaften zugreifen und sie verändern. Mit dem Schlüsselwort „**private**“ kann das verhindert werden.

Nachdem „**private**“ vor den Eigenschaften in den Klassen geschrieben wurde, kann man aus der Hauptklasse nicht auf die Eigenschaften der Objekte nach ihrem Erzeugen mehr direkt zugegriffen werden. Bildlich gesprochen, haben wir also die Eigenschaften unserer Objekte in einer Kapsel verborgen. Man spricht daher in der Objektorientierung auch vom sog. Geheimnisprinzip.

Die Eigenschaften von Klassen sollen aber nicht für immer in ihrer "Kapsel verschlossen" bleiben. Es ist sinnvoll den Zugriff auf die einzelnen Eigenschaften zu kontrollieren. So soll z. B. das Setzen eines Jahrgangs erlaubt sein. Dafür werden in den Klassen die sogenannten Zugriffsmethoden (**set- und get-Methoden**) geschrieben. In der Hauptklasse (dort wo die **main-Methode** zu finden ist) werden diese Methoden aufgerufen.

## Kapitel 9.2: Klassenkarten

Mit einer Klassenkarte wird der Aufbau und Eigenschaften einer Klasse visuell dargestellt.



## **Kapitel 10: Anhang**

Oberfläche entwickeln: Lernpfad: GUI-Programmierung mit dem Java-Editor

[https://www.brichzin.de/unterricht/Lernprogramm\\_javaeditor/index.html](https://www.brichzin.de/unterricht/Lernprogramm_javaeditor/index.html)

Installation Java-Editor und jdk

<https://www.youtube.com/watch?v=FaUtvIg04jA>

<https://www.youtube.com/watch?v=w84Z2GdRyJ4>