

# **PROGRAMMIEREN I**

**MATTHIAS BERG-NEELS**

# VORSTELLUNGSRUNDE

- Name
- Firma
- Guess what?

# ANMERKUNGEN ZUR VORLESUNG

- Skript
  - PDF in public Code Repository (Vorlesung/ProgrammierenSkript.pdf)
- Repositories: <https://gitlab.mubn.de>

# KAPITELÜBERSICHT - PROGRAMMIEREN 1

1. Einführung
2. Grundlagen von Java
3. Datentypen
4. Ausdrücke und Anweisungen
5. Objektorientierung
6. Vererbung
7. Interfaces

# QUELLEN & LITERATURVERZEICHNIS

- **basierend auf dem Skript "Programmieren 1 + 2" von Michael Lang**

---

- BALZERT, HEIDE: Lehrbuch der Objektmodellierung - Analyse und Entwurf. Spektrum Akademischer Verlag, 2. Auflage, 2005. ISBN 3-8274-1162-9
- HÄUSLEIN, ANDREAS: Systemanalyse - Grundlagen, Techniken, Notierungen. VDE Verlag, 2004. ISBN 3-8007-2715-3
- HITZ, M., KAPPEL, G., KAPSAMMER, E. und RETSCHITZEGGER, W.: UML@Work - Objektorientierte Modellierung mit UML 2. dpunkt.verlag, 3., aktualisierte und überarbeitete Auflage, 2005. ISBN 3-89864-261-5
- HOLEY, T., WELTER, G. und WIEDEMANN, A.: Wirtschaftsinformatik. Kiehl Verlag, 2004. ISBN 3-470-52791-1
- RUPP, CHRIS / SOPHIST GROUP: Systemanalyse kompakt. Elsevier Spektrum Akademischer Verlag, 2004. ISBN 3-8274-1509-8
- STAHLKNECHT, P. und HASENKAMP, U.: Arbeitsbuch Wirtschaftsinformatik. Springer Verlag, 4. Auflage, 2006. ISBN 3-540-26361-6
- STAHLKNECHT, P. und HASENKAMP, U.: Einführung in die Wirtschaftsinformatik. Springer Verlag, 10. Auflage, 2002. ISBN 3-540-41986-1
- ZUSER, W., BIFFL, S., GRECHENING, T. und KÖHLE, M.: Software Engineering mit UML und dem Unified Process. Pearson Studium, 2001. ISBN 3-8273-7027-2

# **KAPITEL 1**

# **EINFÜHRUNG**

# ÜBERSICHT

1. **Einführung**
2. Grundlagen von Java
3. Datentypen
4. Ausdrücke und Anweisungen
5. Objektorientierung
6. Vererbung
7. Interfaces

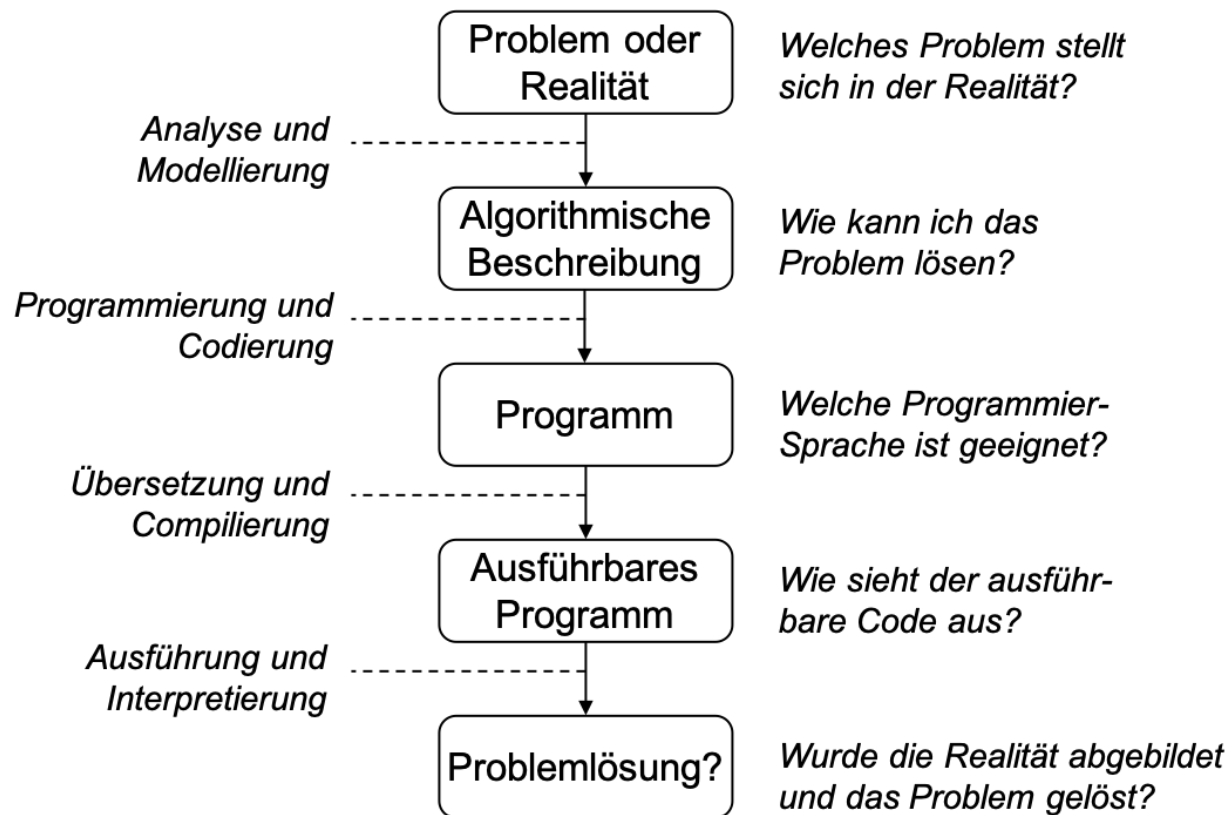
# LERNZIELE

- Sie können den Begriff Algorithmus definieren
- Sie kennen die Eigenschaften und Bestandteile von Algorithmen
- Sie können die Grundbegriffe der Programmierung nennen und einsetzen
- Sie kennen unterschiedliche Darstellungsformen von Algorithmen
- Sie können einfache Algorithmen in Form von Pseudocode, Programmablaufplänen und Struktogrammen darstellen



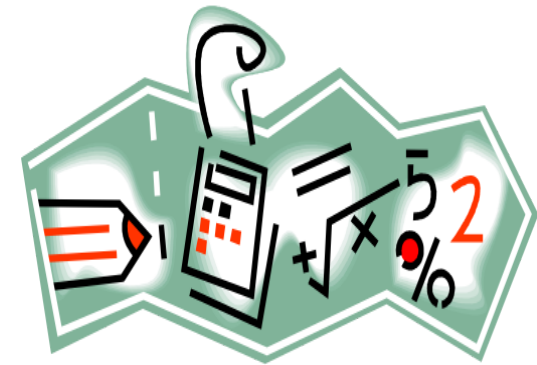
# ZIEL DER PROGRAMMIERUNG

Umsetzung eines gegebenen oder selbstentwickelten Algorithmus in ein lauffähiges Computerprogramm



# BEISPIELE FÜR ALGORITHMEN

- Bedienungsanleitungen
- Bauanleitungen
- Kochrezepte
- mathematische Problemstellungen
- Such- und Sortieralgorithmen



# BEISPIEL: EIN KOCHREZEPT

## Zutaten

- 500 g Hackfleisch vom Rind
- 2 große Zwiebeln, fein gehackt
- 4 Stangen Staudensellerie, fein gehackt
- 150 g Speck, fein gehackt
- 2 Zehen Knoblauch, fein gehackt
- 400 g geschälte Tomaten
- 300 ml Fond (Bratenfond)
- 1 Paprikaschote, rot, klein gewürfelt
- 1 Gewürznelke
- 1 Lorbeerblatt
- ½ TL Oregano
- ½ TL Muskat (gemahlen)
- 3 EL Öl (Oliven)
- 40 g Butter
- 150 ml Wein, rot, trocken
- 1 Chilischote, zerkleinert, getrocknet

## Zubereitung

Butter und Öl in einer Pfanne (besser Bräter) erhitzen. Zwiebeln, Sellerie, Speck und Paprika gut anbraten. Die Sachen aus der Pfanne nehmen und zur Seite stellen. Den Knoblauch leicht anrösten und dann das Hackfleisch dazugeben, alles gut anbraten. Die Sachen wieder dazugeben und alles ca. 10 Min. köcheln lassen. Jetzt alle übrigen Zutaten und Gewürze dazugeben und das Ganze ca. 45 Min. ( bei geschlossenem Deckel ) köcheln lassen (gelegentlich abschmecken und ggf. nachwürzen). Über die fertig gegarten Spaghetti geben.

# PROBLEME BEI DIESEM BEISPIEL

- Zutaten sind bereits vorbereitet
- Spaghetti fehlen bei Zutaten
- Beschreibung für Zubereitung der Spaghetti fehlt
- ungenaue Aussagen
  - fein gehackt
  - klein gewürfelt
  - erhitzen
  - gut anbraten
  - leicht anrösten
  - ...
- Pfanne (besser Bräter)

KURZ: Die Beschreibung lässt Raum für individuelle Entscheidungen und Interpretationen.

# VERBESSERUNGSMÖGLICHKEITEN DES REZEPTS

- komplette Eliminierung von individuellen Interpretationsspielräumen
- vollständige Beschreibung der Arbeitsschritte inkl. Vorbereitung und der Kochanweisung für die Spaghetti
- vollständige Angabe der Zutaten präzise Angaben bei den Aussagen
  - fein gehackt  $\Rightarrow$  gewürfelt,  $2\text{ mm} \leq \text{Kantenlänge} \leq 3\text{ mm}$  (Zwiebeln und Speck)
  - fein gehackt  $\Rightarrow$  gewürfelt,  $0,8\text{ mm} \leq \text{Kantenlänge} \leq 1\text{ mm}$  (Knoblauch)
  - klein gewürfelt  $\Rightarrow 4\text{ mm} \leq \text{Kantenlänge} \leq 6\text{ mm}$
  - erhitzen  $\Rightarrow 120\text{ °C} < \text{Temperatur} < 125\text{ °C}$
  - gut anbraten  $\Rightarrow$  Farbe der Zwiebeln entspricht dem RGB-Wert CC3300
  - leicht anrösten  $\Rightarrow$  Farbe des Knoblauchs entspricht dem RGB-Wert CC6600

# ALGORITHMUS: DEFINITION

## DUDEN

*Al|go|rith|mus, der; , ...men [mlat. algorismus = Art der indischen Rechenkunst, in Anlehnung an griech. arithmós = Zahl entsteht aus dem Namen des pers.- arab. Mathematikers Al-Hwarizmi, gest. nach 846] (Math., Datenverarb.): Verfahren zur schrittweisen Umformung von Zeichenreihen; Rechengvorgang nach einem bestimmten [sich wiederholenden] Schema.*

## INFORMATIK

*Ein Algorithmus ist eine präzise (d.h. in einer festgelegten Sprache abgefasste) endliche Beschreibung eines allgemeinen Verfahrens unter Verwendung ausführbarer elementarer (Verarbeitungs-)Schritte.*

# ALGORITHMUS: EIGENSCHAFTEN

## TERMINIERUNG

- bricht nach endlich vielen Schritten ab
- 

## DETERMINISMUS

- legt die „Wahlfreiheit“ fest
  - deterministischer Ablauf
    - legt eindeutige Vorgabe der Schrittfolge der auszuführenden Schritte fest
  - determiniertes Ergebnis
    - wird immer dann geliefert, wenn bei vorgegebener Eingabe ein eindeutiges Ergebnis geliefert wird - auch bei mehrfacher Durchführung mit denselben Eingabeparametern
-

# ALGORITHMUS: BESTANDTEILE

- elementare Operationen (Ausdrücke und Anweisungen)  
*Berechne 5 plus 7*
- sequenzielle Ausführung  
*Berechne 10 minus 3, dann multipliziere das Ergebnis mit 4*
- parallele Ausführung  
*Du rechnest Aufgabe 1 und ich rechne Aufgabe 2*
- bedingte Ausführung  
*Wenn Du Aufgabe 1 gelöst hast, dann beginne mit Aufgabe 2*
- Schleife  
*Rechne Aufgabe 1, bis Du das richtige Ergebnis bekommst*
- Unterprogramm  
*Rechne Aufgabe 1 anhand der Lösung auf Seite 106*
- Variablen und Konstanten



# GRUNDBEGRIFFE DER PROGRAMMIERUNG

## Ausdruck

- Kombination von Operanden und Operatoren als "Vorschrift" zur Berechnung eines Werts
- liefert immer einen Wert (Ergebniswert) ab
- Beispiel:  $1 / x$

## Anweisung

- Kombination von Ausdrücken und Methoden als "Vorschrift" zur Ausführung einer Aktion
- Beispiele:
  - $y = 1 / x$  Wertzuweisung
  - `print(x)` Ausgabeanweisung (Methodenaufruf "Drucke x")

# GRUNDBEGRIFFE DER PROGRAMMIERUNG

## Sequenz

- bildet eine zeitliche Abfolge von Anweisungen
- einzelne Schritte werden durchnummeriert oder es wird zum Abschluss der Sequenz ein Semikolon gesetzt

## Bedingte Anweisung

- es werden Bedingungen auf Ihre Richtigkeit geprüft
- für wahre und falsche Aussagen in der Bedingung können unterschiedliche Anweisungen ausgeführt werden

## Schleifen

- bestimmte Anweisungen werden wiederholt, bis eine definierte Endbedingung erfüllt wird
- Unterscheidung in drei Schleifenarten

# GRUNDBEGRIFFE DER PROGRAMMIERUNG

## Unterprogramme

- beinhaltet einen Teilalgorithmus
- dieser Teilalgorithmus kann in mehreren Algorithmen wieder verwendet werden

## Variablen

- „Platzhalter“ für einen konkreten Wert
- sind von einem bestimmten Datentyp können ihren Wert ändern

## Konstanten

- haben einen festen Wert
- sind von einem bestimmten Datentyp
- können ihren Wert NICHT ändern

# DARSTELLUNGSFORMEN VON ALGORITHMEN

## Pseudocode

- nahe an den Konstrukten verbreiteter Programmiersprachen
- Verwendung spezieller englischer Begriffe aus dem Alltag
- Begriffe haben eine festgelegte Bedeutung

## Programmablaufpläne

- genormt nach DIN 66001
- Ursprung in der linearen Programmierung
- nur für kleinere Programme geeignet (Übersichtlichkeit)

## Nassi-Schneiderman-Diagramme (Struktogramme)

- Entstehung 1973
- Darstellung genormt nach DIN 66261 im Jahr 1985
- überwiegender Einsatz in der prozeduralen Programmierung

# PSEUDOCODE

## Sequenz

- Alternative 1: Schritte werden durchnummeriert: 1, 2, 3, ...
- Alternative 2: Abschluss der Sequenz durch Semikolon
- Vorteil Alternative 1: Verfeinerung einzelner Schritte: 2.1, 2.2, ...

## Bedingte Anweisung

- es werden Bedingungen auf Ihre Richtigkeit geprüft
- Alternative 1: falls Bedingung dann Schritt
- Alternative 2: falls Bedingung dann Schritt A sonst Schritt B

## Schleifen

- kopfgesteuert: solange Bedingung wahr führe aus Schritte
- fußgesteuert: wiederhole Schritte bis Bedingung wahr
- Zählschleife: wiederhole für Zahlenbereich Arbeitsschritte

# BEISPIEL: PSEUDOCODE

## SEQUENZ

1. Koche Wasser
2. Gib Kaffeepulver in Tasse
3. Fülle Wasser in Tasse

## BEDINGTE ANWEISUNG

**falls** Ampel rot oder gelb  
    **dann** stoppe  
    **sonst** fahre weiter

**falls** Ampel ausgefallen  
    **dann** fahre vorsichtig weiter  
    **sonst falls** Ampel grün  
        **dann** fahre weiter  
    **sonst** stopp

## SCHLEIFEN

**solange** Liste nicht erschöpft  
    **führe aus**

        Gib nächste Zahl aus der Liste aus

**wiederhole**

    Gib nächste Zahl aus der Liste aus  
    **bis** Liste erschöpft

**wiederhole für** 5 bis 10

    Gib nächste Zahl aus der Liste aus



# STRUKTOGRAMME

## Sequenz

Eingabe: 2 Zahlen ohne Kommastellen

Addiere die beiden Zahlen

Ausgabe: Ergebnis der Addition

## Kopfgesteuerte Schleife

Zähler auf 0 setzen

Solange Zähler kleiner 100 ist

Zähler um 1 erhöhen

## Bedingte Anweisung

Eingabe: 2 Zahlen ohne Kommastellen

Zahl 1 > Zahl 2

Ausgabe: Zahl 1 ist  
größer als Zahl 2

Ausgabe: Zahl 2 ist  
größer als Zahl 1

## Fußgesteuerte Schleife

Zähler auf 0 setzen

Zähler um 1 erhöhen

bis Zähler gleich 100 ist

## Mehrfachverzweigung

Eingabe: 2 Zahlen und 1 Operator

		Operator ist
+	-	
Ergebnis = Zahl 1 + Zahl 2	Ergebnis = Zahl 1 - Zahl 2	Operation nicht möglich

Ausgabe: 2 Zahlen, Operator, Ergebnis

## Zählschleife

Für Zähler von 1 bis 100

Ausgabe: Zähler

Ausgabe: „Unser Rechner kann Zählen“

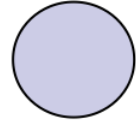




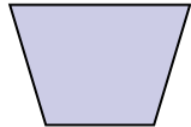
# PROGRAMMABLAUFPLAN



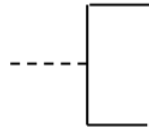
**Allgemeine Verarbeitung**  
(einschl. Ein- und Ausgabe)



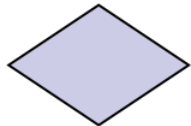
**Verbindungsstelle**



**manuelle Verarbeitung**  
(einschl. Ein- und Ausgabe)



**Bemerkung**  
(erläuternder Text)



**Verzweigung**

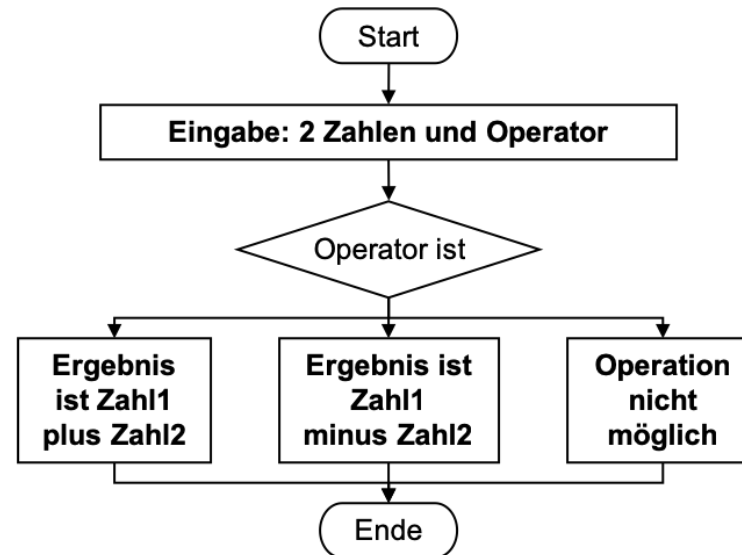
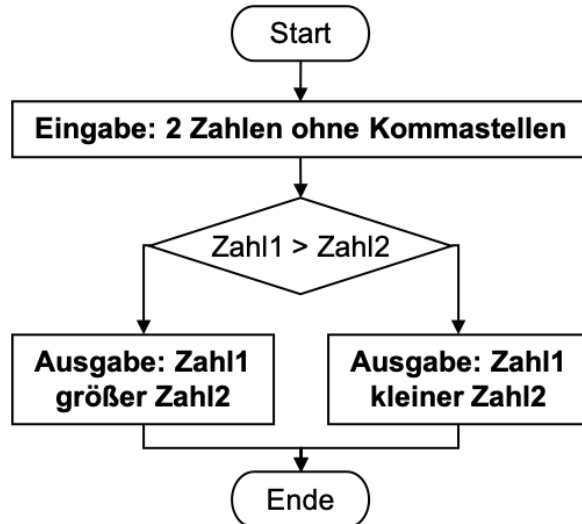
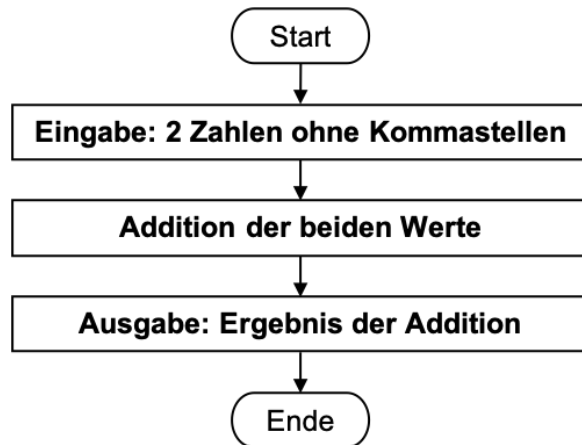


**Verbindung**  
(Verarbeitungsfolge)



**Grenzstelle (zur Umwelt)**

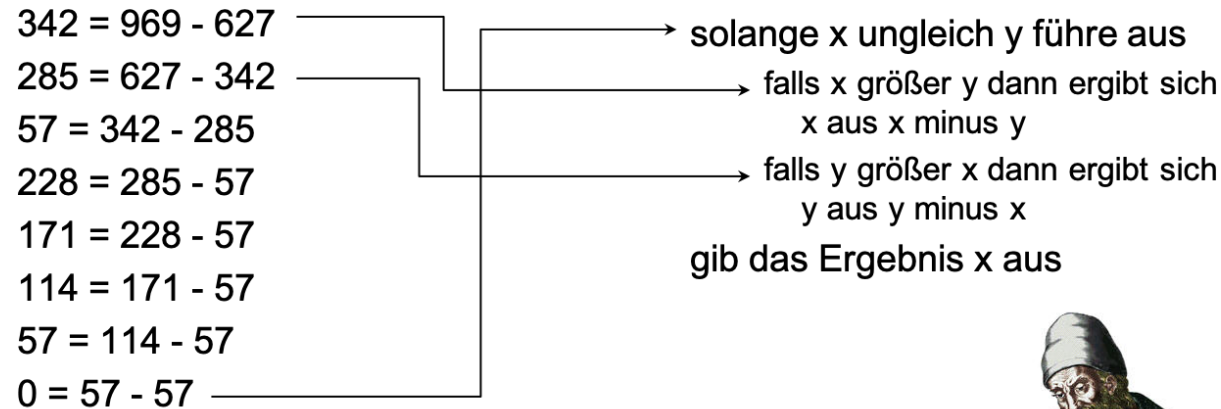
# BEISPIEL: PROGRAMMABLAUFPLAN



# BEISPIEL: MATHEMATISCHE PROBLEMSTELLUNG

gesucht sei  $\text{ggT}(969,627)$  durch Anwendung des Euklidschen Algorithmus

allgemeingültige Formulierung mit den Variablen  $x$  und  $y$ :  $\text{ggT}(x,y)$



**Übung:** Erstellen Sie zum beschriebenen des euklidschen Algorithmus ein Struktogramm und einen Programmablaufplan.

# **KAPITEL 1**

# **EINFÜHRUNG**

## **KONTROLLFRAGEN & ÜBUNGEN**

# KONTROLLFRAGEN

1. Wie würden Sie den Begriff Algorithmus im Rahmen der Informatik definieren?
2. Beschreiben Sie die Bestandteile sowie die Eigenschaften von Algorithmen!
3. Nennen Sie fünf Grundelemente der Programmierung!
4. Welche Möglichkeiten stehen Ihnen zur Verfügung, um Algorithmen grafisch darzustellen?
5. Was ist ein wesentlicher Nachteil der Programmablaufpläne?
6. Beschreiben Sie die Darstellungsform „Pseudocode“!

# ÜBUNGEN

1. Beschreiben Sie den euklidischen Algorithmus zur Berechnung des größten gemeinsamen Teilers als Struktogramm und Programmablaufplan!
2. Beschreiben Sie einen Algorithmus zur Berechnung der Fakultät! Nutzen Sie dazu den Pseudocode.
3. Beschreiben Sie im Pseudocode einen Algorithmus zur Bestimmung der Fibonacci Folge 1 – 1 – 2 – 3 – 5 – 8 – 13 – 21!
4. Beschreiben Sie in einem Struktogramm das Sieb des Eratosthenes zur Bestimmung von Primzahlen.

# **KAPITEL 2**

## **GRUNDLAGEN VON JAVA**



# ÜBERSICHT

1. Einführung
2. **Grundlagen von Java**
3. Datentypen
4. Ausdrücke und Anweisungen
5. Objektorientierung
6. Vererbung
7. Interfaces

# LERNZIELE

- Sie können die Eigenschaften der Programmiersprache Java beschreiben
- Sie kennen die Aufgaben von Compiler, Linker und Interpreter
- Sie können das Zusammenspiel von Bytecode und der Java Virtual Machine erläutern
- Sie können die wesentlichen Java-Tools benennen und ihre Aufgabe beschreiben
- Sie können das Paketkonzept in Java erläutern
- Sie kennen die wesentlichen Systemvariablen im Umfeld von Java
- Sie können Eclipse als Java-Entwicklungsumgebung einsetzen

# VORBEREITUNG

- [prüfen] Installieren Sie das Java Developer Kit, falls notwendig. [JDK Download von Oracle](#)
- Aktivieren Sie eine Studierenden-Lizenz und installieren Sie IntelliJ [Jetbrains Free License for faculty members - DHBW-Email benötigt](#)
- [optional] Registrieren Sie sich auf <https://gitlab.mubn.de> und beantragen Sie unter "Groups" > "explore Groups" für die Gruppe Ihres Kurses den Zugang

# KLASSIFIZIERUNG VON SOFTWARE

- Programme zur Steuerung der Verarbeitungsprozesse, der Übertragungsprozesse und der Speicherungsprozesse in Computern
- unterschiedliche Softwarearten
  - Systemsoftware
    - grundlegende Dienste für andere Programme
    - Steuerung des Computersystems (Hardware)
    - Ablaufsteuerung anderer Programme
  - Entwicklungssoftware
    - Erstellung und Modifikation von Programmen
    - Übersetzungsprogramme für Programmiersprachen
  - Anwendungssoftware
    - Programme zur Verarbeitung der Daten
    - Unterhaltungssoftware
    - Spiele

# EIGENSCHAFTEN VON JAVA

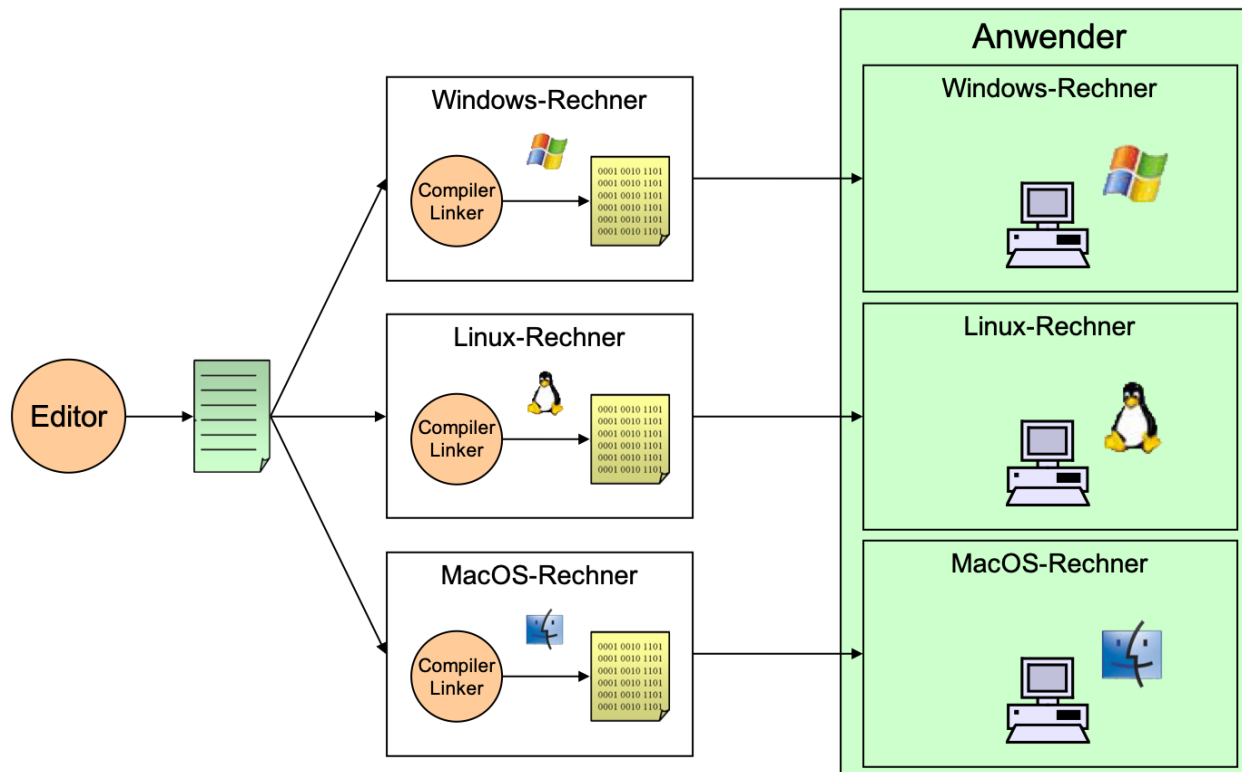
- vollständig objektorientiert
  - ohne prozedurale Altlasten
- unkompliziert - einfach und leicht erlernbar
  - Syntax ähnlich zu C, C++
  - Beschränkung auf das Notwendigste
  - keine Pointer
  - keine Header-Dateien
  - keine Präprozessor-Anweisungen
  - keine Mehrfachvererbung
- plattformunabhängig (architekturneutral)
  - übersetzte Java-Programme (Bytecode) sind auf jeder javafähigen Plattform ausführbar
  - fest definierter Wertebereich für Zahlen

# EIGENSCHAFTEN VON JAVA

- sicher
  - keine direkten Speicherzugriffe mit \* Pointerarithmetik
  - strenge Typüberprüfung
  - keine Programmierung mit Sprachverletzung
- robust
  - keine Rechnerabstürze durch Programmierfehler
  - Überprüfung der Speicherzugriffe
  - Ausnahmeroutinen zur Fehlerbehandlung
- multithreaded
  - parallele Ausführung von Programmteilen
- internetfähig
  - Java-Applets sind über das Internet verteilbar und können lokal auf dem Rechner ausgeführt werden

# **UMWANDLUNG VON QUELLCODE IN MASCHINENSPRACHE**

# COMPILER





# FUNKTIONSWEISE EINES COMPILER

- übersetzt ein Computerprogramm aus einer Quellsprache in ein semantisch äquivalentes Programm einer Zielsprache
- Aufbau eines Compilers in 2 Phasen
  - Analysephase
    - lexikalische Analyse zerteilt Code in zusammengehörende Token
    - syntaktische Analyse überprüft auf formale Richtigkeit
    - semantische Analyse überprüft die logischen Rahmenbedingungen
  - Synthesephase
    - Zwischencodeerzeugung liefert die Basis für die Optimierung
    - Programmoptimierung auf Basis des maschinennahen Zwischencodes
    - Codegenerierung erzeugt den Programmcode der Zielsprache

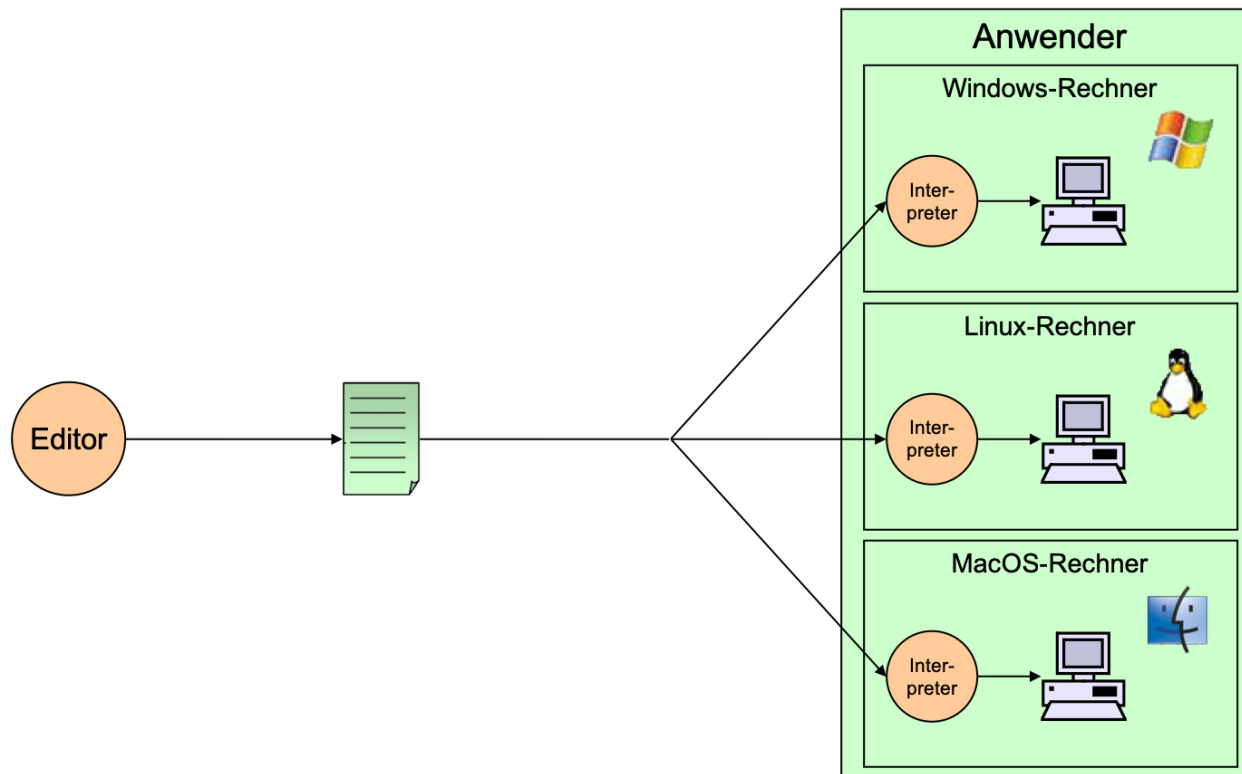
## **ARTEN VON COMPILERN**

- Native Compiler erzeugt Code für die Plattform, auf der er läuft
- Cross-Compiler erzeugt Code für andere Plattformen
- One-pass-Compiler erzeugt den Code in einem Durchlauf
- Multi-pass-Compiler erzeugt den Code in mehreren Durchläufen

# LINKER

- stellt einzelne Programmmodule zu einem ausführbaren Programm zusammen
- fügt benötigten Code aus Funktionsbibliotheken zum Code des Hauptprogramms hinzu
- statisches Linken
  - beim kompilieren werden benötigte Codings aus Funktionsbibliotheken dem Coding des Hauptprogramms hinzugefügt
- dynamisches Linken
  - zur Laufzeit werden die Codings aus Funktionsbibliotheken o.ä. zum Hauptprogramm hinzugelinkt
    - DLL-Konzept von Windows
    - auch bei Java findet dynamisches Linken statt

# INTERPRETER



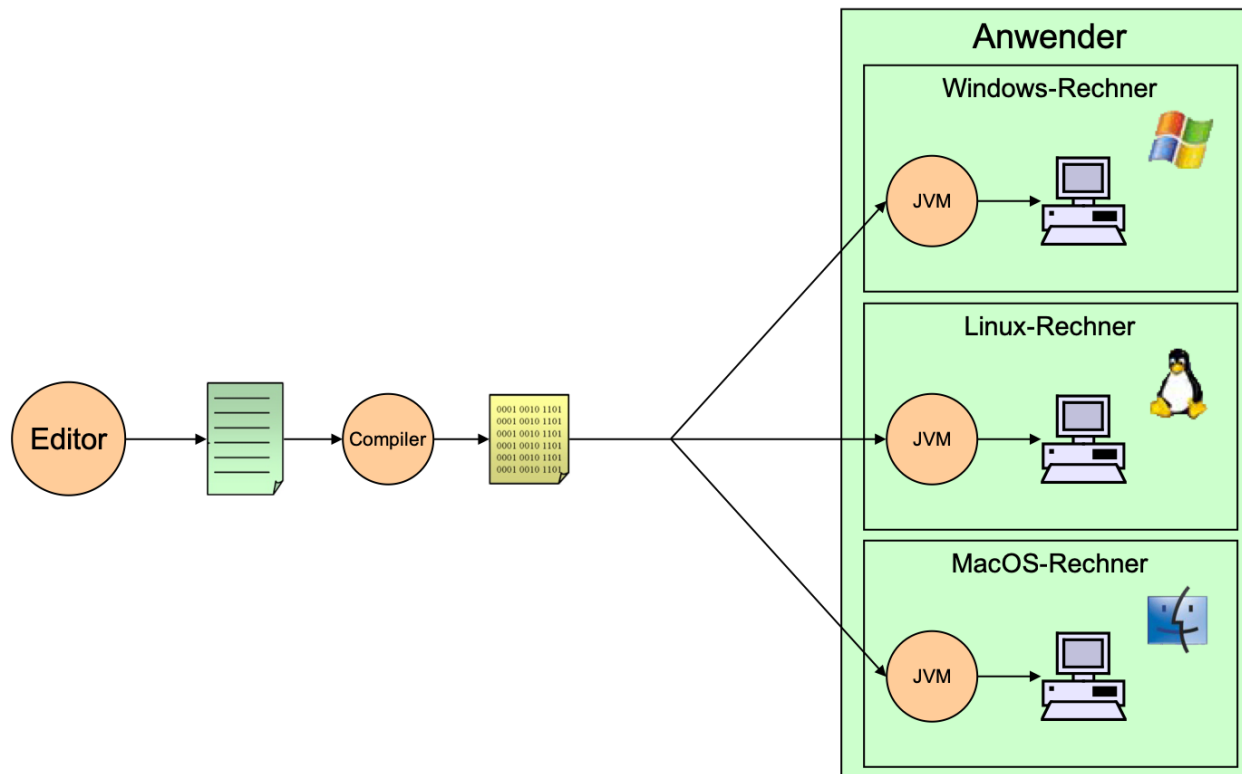
# FUNKTIONSWEISE EINES INTERPRETERS

- übersetzt den Quellcode nicht in eine direkt ausführbare Datei
- liest den Quellcode ein, analysiert diesen und führt ihn dann aus
- die Analyse erfolgt also zur Laufzeit des Programms
- auf jeder Rechnerarchitektur lauffähig
- deutlich langsamer als kompilierte Programme
- Möglichkeiten zur Steigerung der Geschwindigkeit
  - Just-In-Time-Compiler
    - zur Laufzeit wird der Quellcode in einen Maschinencode übersetzt
    - Maschinencode wird direkt vom Prozessor ausgeführt
    - mehrfach durchlaufene Programmteile müssen nur ein Mal übersetzt werden
    - nur auf einer bestimmten Rechnerarchitektur lauffähig
  - Bytecode-Interpreter
    - Quellcode wird zur Laufzeit in sog. Bytecode übersetzt
    - Bytecode wird von einem Interpreter (Virtual Machine) ausgeführt
    - Bytecode kann auf verschiedenen Plattformen ausgeführt werden



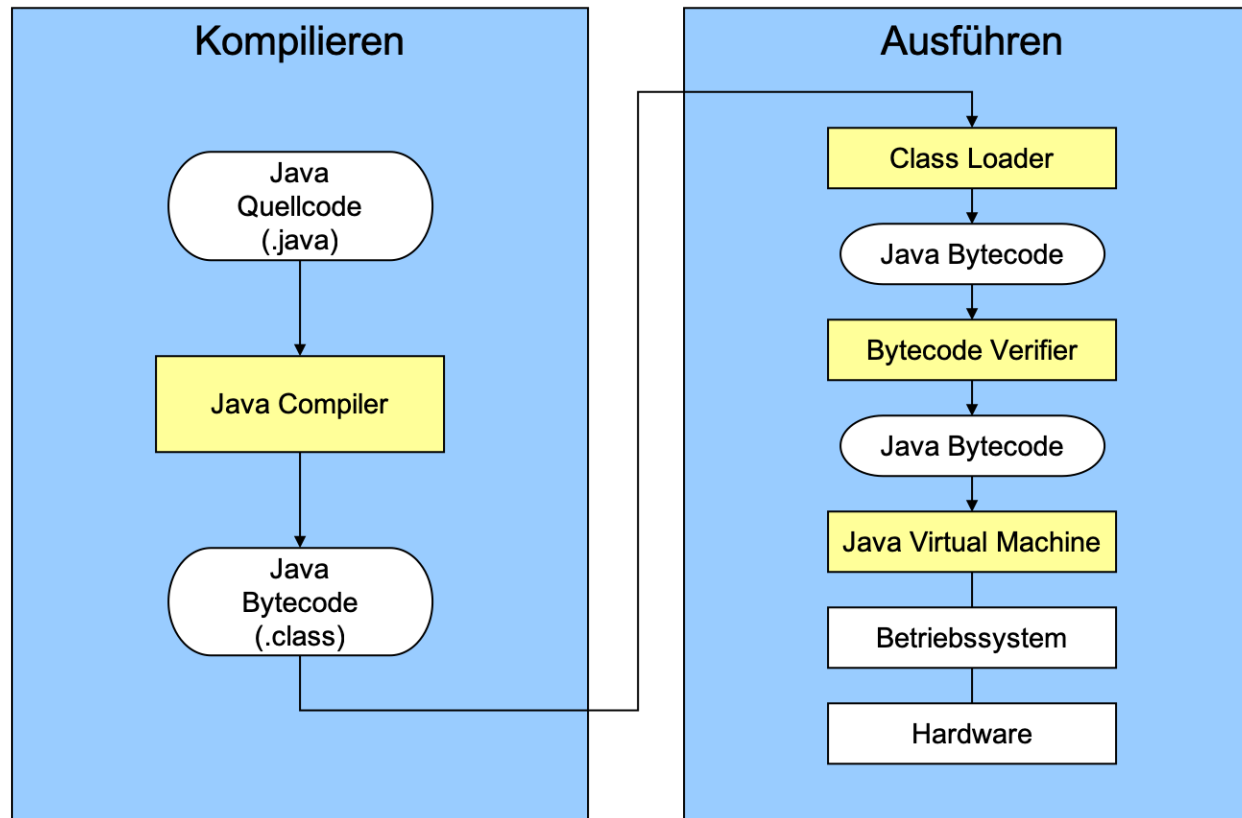
# **JAVAQUELLCODE ZU MASCHINENSPRACHE**

# PLATFORMUNABHÄNGIGKEIT DURCH BYTECODE





# BYTECODE UND VIRTUAL MACHINE



# WESENTLICHE JAVA-TOOLS

Java-Compiler (javac.exe)

Java virtual Machine (java.exe)

Javadoc

- dient der automatischen Erstellung von Dokumentationen

Class Loader

- lädt Java-Klassen in den Arbeitsspeicher
- bereitet Ausführung von Java-Applikationen vor

Bytecode Verifier

- Prüfung auf syntaktische Korrektheit
- Überprüfung der Klassenhierarchie
- Überprüfung jeder Methode auf strukturelle Gültigkeit
- Datenflussanalyse, um u.a. Typfehler zu vermeiden

# WICHTIGE SYSTEMVARIABLEN

## PATH

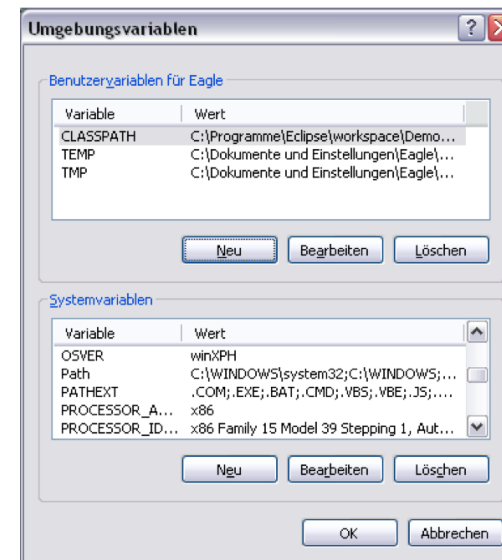
- Verzeichnis, in dem sich die JAVA-Tools befinden

## CLASSPATH

- Verzeichnisse, in denen nach Klassen und Paketen gesucht werden soll

## JAVA\_HOME

- Verzeichnis, in dem das J2SDK installiert wurde



# BEISPIEL: HELLO WORLD ÜBER DIE KONSOLE

1. anlegen einer Datei "HelloWorld.java" mit folgendem Quellcode

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

1. Quellcode zu Bytecode kompilieren

```
javac HelloWorld.java
```

1. Ausführen des kompilierten Bytecode in Java Virtual Machine (JVM)

```
java HelloWorld
```

# JAVA - KOMMENTARE IM QUELLCODE

## Einzeiliger Kommentar

```
// single line comment -
```

## Blockkommentar (Mehrzeiliger Kommentar)

```
/*  
multi line comment  
starting with the signs  
  
ending with the signs  
*/
```

## Beispiel

```
// HelloWorld class as first small programming example  
class HelloWorld {  
  
    // main method as starting point in Code  
    public static void main (String[] args){  
        /*  
        Implementation of main method  
        prints out "Hello to the Java World"  
        to the console.  
        */  
        System.out.println("Hello to the Java World");  
    }  
}
```



# EXKURS: JAVADOC

## GENERIERUNG VON DOKUMENTATION AUS JAVAQUELLCODE UND KOMMENTAREN

- Generiert Dokumentation anhand des Quellcodes
- Kommentare mit spezieller Formatierung ("/\*\* ") werden berücksichtigt
- Resultat: Dokumentation im HTML-Format

Beispiel:

```
/** HelloWorld class as first programming example
 * @author Matthias Berg-Neels
 * @version 1.0
 * @since 1.0
 */
class HelloWorld {

    /** main method as starting point for program start
     * @param args String array for parameters from the console
     */
    public static void main (String[] args){

        System.out.println("Hello to the Java World");
    }
}
```

# GENERIERUNG DER DOKUMENTATION

```
javadoc HelloWorld.java -html5 -author -version
```

Ergebnis:

[PACKAGE](#) [CLASS](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)    [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

SEARCH:

### Class HelloWorld

java.lang.Object  
HelloWorld

---

```
public class HelloWorld
extends java.lang.Object
```

HelloWorld class as first programming example

Since:  
1.0

Version:  
1.0

Author:  
Matthias Berg-Neels

#### Constructor Summary

[Constructors](#)

Constructor	Description
<code>HelloWorld()</code>	

#### Method Summary

[All Methods](#) [Static Methods](#) [Concrete Methods](#)

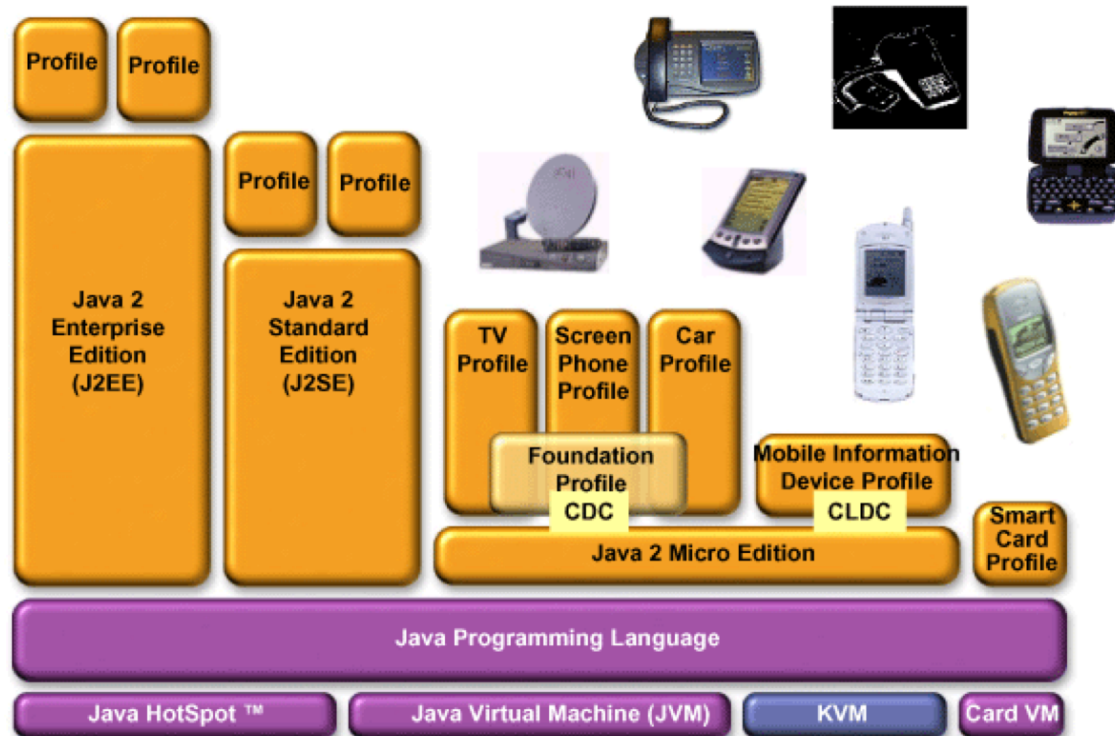
Modifier and Type	Method	Description
static void	<code>main(java.lang.String[] args)</code>	main method as starting point for program start



# DAS PAKETKONZEPT IN JAVA

- Möglichkeit zur Strukturierung bzw. sinnvollen Sortierung von Klassen
- Pakete sind somit Sammlungen von Klassen
- die Klassen verfolgen einen gemeinsamen Zweck
- jede Klasse ist genau einem Paket zugeordnet
- Paketnamen können aus mehreren Teilen bestehen und hierarchisch aufgebaut sein (vergleichbar mit der Ordnerstruktur im Windows-Explorer)
- eine Klasse kann eindeutig über das Paket und ihren Namen identifiziert werden

# FUNKTIONSUMFANG DER JAVA 2™ PLATFORM



# FUNKTIONSUMFANG DER JAVA 2™ PLATFORM

- Java 2 Standard Edition beinhaltet das Software Development Kit mit der Standard API zur Entwicklung von Java-Applikationen
- Java 2 Enterprise Edition umfasst neben der J2SE weitere Packages zur serverseitigen Programmierung (Enterprise Java Beans, Servlets, JSP, Java-Mail-API, etc.)
- Java 2 Micro Edition stellt eine funktional kleinere Laufzeitumgebung für mobile Endgeräte (PDAs, Handys, Navigationssysteme, etc.) dar
- Connected Device Configuration (CDC) beinhaltet die komplette JVM und ist in mobilen Systemen integriert
- Connected Limited Device Configurations (CLDC ) ist eine J2ME-Bibliothek zur Abdeckung gerätespezifischer Funktionen; ermöglicht die Zusammenarbeit verschiedener Geräte der gleichen Kategorie
- KVM ist die kleinste Laufzeitumgebung und wird für den Einsatz auf Geräten mit beschränkter Speicherkapazität und CPU-Leistung verwendet
- Java Card APIs definieren eine minimale Laufzeitumgebung auf SmartCards

# ENTWICKLUNGSUMGEBUNG

## INTELLIJ IDEA

- Entwickler: [Jetbrains](#)
- Integrated Development Environment (IDE) für unterschiedliche Programmierspachen und Umgebungen
  - **Java**, Javascript (Node.js), Android, Kotlin, ...
- freie Community Edition
- seit Version 9.0: kostenpflichtige Version (sehr beliebt)
- seit Version 14.0: Decompiler für Java Bytecode
- Erweiterbarkeit durch Plug-Ins
- verschiedene Ableger für weitere Programmiersprachen
  - z.B. Webstorm für PHP

# **KAPITEL 2**

# **GRUNDLAGEN VON JAVA**

## **KONTROLLFRAGEN**

# KONTROLLFRAGEN

1. Beschreiben Sie die wesentlichen Eigenschaften von Java!
2. Beschreiben Sie die Funktionsweise eines Compilers!
3. Welche Aufgaben übernimmt der Linker?
4. Nennen Sie unterschiedliche Interpreter-Arten!
5. Welche Rolle spielen Compiler und Interpreter im Umfeld der Programmiersprache Java?
6. Beschreiben Sie den Prozess von der Erstellung des Quellcodes bis zur Ausführung des Programms in der Programmiersprache Java!
7. Nennen Sie drei wesentliche Java-Tools und beschreiben Sie kurz deren Aufgaben!
8. Beschreiben Sie das Paketkonzept der Programmiersprache Java!
9. Welche wesentlichen Systemvariablen kennen Sie im Java-Umfeld?
10. Beschreiben Sie die wesentlichen Unterschiede der verschiedenen Editionen im Rahmen der Java 2™ Platform!
11. Wozu werden die einzelnen Systemvariablen benötigt?

# **KAPITEL 3**

## **DATENTYPEN**

# ÜBERSICHT

1. Einführung
2. Grundlagen von Java
3. **Datentypen**
4. Ausdrücke und Anweisungen
5. Objektorientierung
6. Vererbung
7. Interfaces



# LERNZIELE

- Sie kennen die unterschiedlichen einfachen Datentypen in Java
- Sie kennen die Wertebereiche der jeweiligen Datentypen
- Sie können Variablen und Konstanten in Java deklarieren
- Sie kennen die unterschiedlichen Literale
- Sie kennen unterschiedliche Standard-Escape-Sequenzen und können diese einsetzen
- Sie kennen die Konvertierungsvorschriften bezüglich der einfachen Datentypen
- Sie können den Aufbau von Arrays beschreiben
- Sie können Arrays in Java deklarieren
- Sie kennen den theoretischen Umgang mit Referenzdatentypen
- Sie kennen die speziellen Referenzdatentypen Array und String

# ARTEN VON DATENTYPEN



# ARTEN VON DATENTYPEN

- bool'sche Typen sind Wahrheitswerte: True und False
- numerische Typen
  - Byte, Short, Integer und Long: Menge der ganzen Zahlen
  - Double und Float: Menge der reellen Zahlen
- alphanumerisch
  - Char: Menge der vereinbarten Schriftzeichen, z.B. ASCII-Zeichensatz
- komplexe Datentypen
  - String: Verkettung von Character-Werten  $\Rightarrow$  Zeichenketten
  - Array: Definition eines n-dimensionalen Feldes
  - Referenzdatentypen: Individuell definierte Datentypen

# WERTEBEREICHE UND SPEICHERVERBRAUCH

Typname	Länge in Byte	Wertebereich
boolean	1	<i>true, false</i>
char	2	Alle Unicode-Zeichen
byte	1	$-2^7$ bis $2^{7-1}$ (-128 bis 127)
short	2	$-2^{15}$ bis $2^{15-1}$ (-32.768 bis 32.767)
int	4	$-2^{31}$ bis $2^{31-1}$ (-2.147.483.648 bis 2.147.483.647)
long	8	$-2^{63}$ bis $2^{63-1}$ (-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807)
float	4	$\pm 3.40282347 * 10^{38}$
double	8	$\pm 1.79769313486231570 * 10^{308}$

# DEKLARATION VON VARIABLEN

- Variablen werden durch Datentyp und Variablennamen deklariert

```
Typname Variablenname;
```

- Beispiele für Variablendeklarationen

```
boolean muede;  
int a;
```

- Beispiele für Deklarationen mit Initialisierung

```
char buchstabe = 'a';  
long zahl = 45768;
```

# DEKLARATION VON KONSTANTEN

- Konstanten werden ähnlich wie Variablen deklariert vor den Typnamen wird das Schlüsselwort

```
final gesetztfinal Typname Konstantenname;
```

- Konstanten können nach der Deklaration einmalig mit einem Wert initialisiert werden

```
final double pi = 3.1415926535897932384626433832795;
```

# LITERALE

*Ein Literal (lateinischen littera "Buchstabe") in der Programmiersprache bezeichnet fest definierte Zeichenfolgen zur Darstellung von Werten für Basis-Datentypen.*

# NUMERISCHE LITERALE

## GANZZAHLEN

Literale für die Familie der Integer-Werte (byte, short, int, long) können in Dezimal-, Oktal- oder Hexadezimalform geschrieben werden

- Dezimalliterale bestehen aus den Ziffern 0 bis 9
- Oktalliterale bestehen aus dem Präfix 0 und den Ziffern 0 bis 7
- Hexadezimalliterale bestehen aus dem Präfix 0x, den Ziffern 0 bis 9 und den Buchstaben A bis F bzw. a bis f
- durch das Suffix l bzw. L wird ein Literal vom Typ long erzeugt



# NUMERISCHE LITERALE

## FLIESSKOMMAZAHLEN

Literale für Fließkommazahlen float und double

- bestehen aus einem Vorkommateil, dem Dezimalpunkt, einem Nachkommateil, einem Exponenten und einem Suffix
- Unterscheidung zwischen float und double durch das Suffix f bzw. d
- Exponent wird durch ein e bzw. E eingeleitet
- Vorkomma- oder Nachkommateil darf ausgelassen werden, der Exponent und das Suffix sind optional
- ohne Suffix sind Fließkommazahlen immer vom Typ double

# ALPHANUMERISCHE LITERALE

- besondere Literale für den Datentyp boolean

*true und false*

- 
- Char-Literale werden grundsätzlich in einfache Hochkommata gesetzt
  - Char-Variablen sind prinzipiell 2 Byte lang, da in char Unicode-Werte gespeichert werden können

```
'a';  
'\u2764';
```

- 
- besondere Zeichenliterale stellen die String-Literale dar
  - String-Literale stehen in doppelten Hochkommata

```
"Hier ist eine Zeichenkette"
```

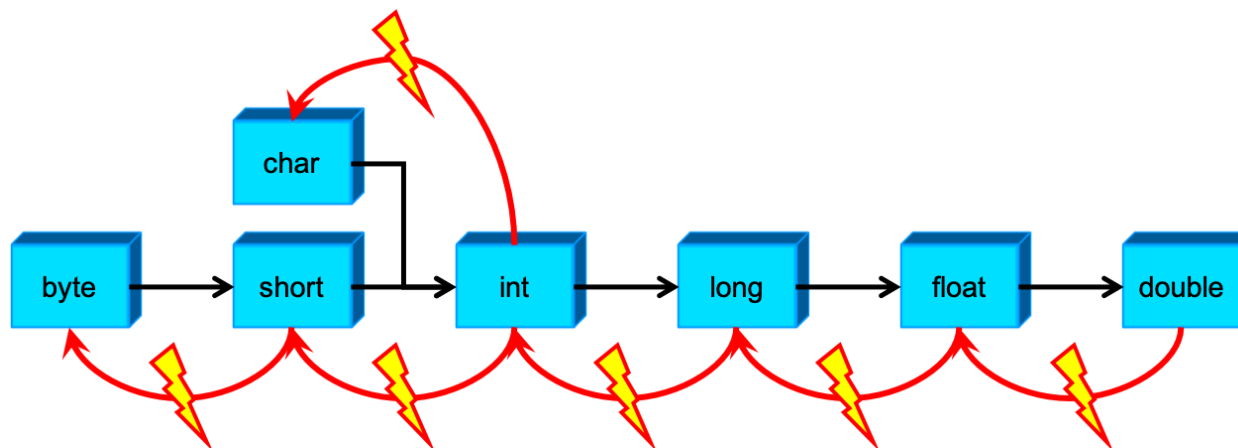
# STANRD-ESCAPE-SEQUENZEN

- Zeichenliterale können Standard-Escape-Sequenzen zur Darstellung von Sonderzeichen beinhalten
- Beispiele:

Zeichen	Bedeutung
<code>\t</code>	Horizontaler Tabulator
<code>\n</code>	Zeilenschaltung (Newline)
<code>\“</code>	Doppeltes Anführungszeichen
<code>\‘</code>	Einfaches Anführungszeichen
<code>\\</code>	Backslash
<code>\uxxxx</code>	Unicodezeichen (xxxx steht für den Hexadezimalwert des Unicodezeichens)

# KONVERTIERUNGSREGELN

- generelle Unterscheidung in erweiternde und einschränkende Konvertierung
- es existiert keine Konvertierungsvorschrift für den Datentyp boolean und für die Konvertierung zwischen einfachen und Referenzdatentypen



# ARRAYS

- Arrays sind (mehrdimensionale) Feldvariablen, die aus mehreren Elementen bestehen
- alle Elemente eines Arrays gehören dem gleichen Datentyp an
- in Java sind Arrays semidynamisch, d.h. ihre Größe kann zur Laufzeit festgelegt, aber danach nicht mehr verändert werden
- Arrays in Java sind Objekte und bieten verschiedene Methoden
- die Elemente eines Arrays sind bei n Elementen von 0 bis n-1 durchnummeriert
- Zugriffe auf einzelne Elemente von Arrays erfolgen über den numerischen Index
- das Attribut length ist vom Typ Integer und gibt die Anzahl der Elemente eines Arrays zurück

# DEKLARATION UND INITIALISIERUNG VON ARRAYS

- die Deklaration eines Arrays entspricht syntaktisch der einer einfachen Variablen
- Unterschied: an den Typnamen oder an den Variablennamen werden eckige Klammern angefügt
- die Initialisierung erfolgt mithilfe des new-Operators oder durch Zuweisung eines Array-Literals
- Beispiele

```
int [] zahl = {1, 2, 3, 4, 5};  
zahl[1] = 7;
```

```
int [][] matrix = new int[3][3];  
matrix[1][2] = 15;
```

```
int [][] matrix = new int[3][];  
matrix[0] = new int[4];
```

```
int [][][] wuerfel = new int[3][3][3];  
wuerfel[2][0][0] = 73;
```

	<b>7</b>			
--	----------	--	--	--

	<b>15</b>	

--	--	--

⋮

		<b>73</b>	

# UMGANG MIT REFERENZDATENTYPEN

- im Gegensatz zu einfachen Datentypen handelt es sich bei Referenzdatentypen um komplexe Typen
- zu den Referenztypen zählen
  - Objekte (näheres im Kapitel 5: Objektorientierung)
  - Strings
  - Arrays
- Strings und Arrays sind besondere Objekte
- sowohl bei Strings als auch bei Arrays kennt der Compiler Literale, die den Aufruf des Operators new überflüssig machen
- bei einfachen Datentypen reicht die Deklaration der Variablen aus, während Referenztypen mittels des new-Operators noch erzeugt werden müssen (Ausnahmen Strings und Arrays)
- Referenztypen stellen lediglich einen Verweis auf Objekte dar



# **KAPITEL 3**

# **DATENTYPEN**

## **KONTROLLFRAGEN**

# KONTROLLFRAGEN

1. Welche unterschiedlichen Datentypen kennen Sie in Java?
2. Wie lassen sich diese Datentypen klassifizieren?
3. Nennen Sie die numerischen Datentypen der Programmiersprache Java!  
Worin liegt der wesentliche Unterschied im Wertebereich?
4. Wie können Variablen und Konstanten in Java deklariert initialisiert werden?
5. Was ist der Unterschied zwischen einer Konstanten und einer Variablen?
6. Erläutern Sie den Begriff „Literal“ am Beispiel der numerischen Literale!
7. Was sind Escape-Sequenzen?
8. Beschreiben Sie die Konvertierungsregeln in Java bzgl. Erweiternder und einschränkender Konvertierungen für einfache Datentypen!
9. Was ist ein Array und wie ist dieses aufgebaut?
10. Welche Aufgabe erfüllt das Attribut length bei einem Array?
11. Worin unterscheidet sich die Deklaration eines Arrays von der Deklaration einer einfachen Variablen?
12. Was sind Referenzdatentypen?

# **KAPITEL 4**

## **AUSDRÜCKE & ANWEISUNGEN**

# ÜBERSICHT

1. Einführung
2. Grundlagen von Java
3. Datentypen
4. **Ausdrücke und Anweisungen**
5. Objektorientierung
6. Vererbung
7. Interfaces

# LERNZIELE

- Sie kennen die Begriffe Ausdrücke und Anweisungen
- Sie kennen die unterschiedlichen Arten von Operatoren
  - arithmetisch
  - relational
  - logisch
  - Zuweisung
  - sonstige
- Sie kennen die Verzweigungsmöglichkeiten mit der IF- und der SWITCH-Anweisung
- Sie kennen die unterschiedlichen Schleifentypen
  - kopfgesteuert
  - fußgesteuert
  - zählend
- Sie kennen die break- und continue-Anweisung