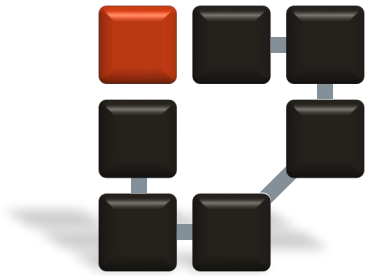
Informatik 1 für Nebenfachstudierende Grundmodul

Java - Einführung

Kai-Steffen Hielscher
Folienversion: 03. Dezember 2019



Informatik 7
Rechnernetze und
Kommunikationssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Inhaltsübersicht

- Kapitel 3 - Java
  - **Einführung**
  - Lexikalische Struktur
  - Datentypen und Variablen

Algorithmen

- Ein **Algorithmus** ist eine eindeutige, endliche Beschreibung eines allgemeinen, endlichen Verfahrens zur schrittweisen Ermittlung gesuchter Größen (Output) aus gegebenen Größen (Input)
  - Algorithmen sind eine der wesentlichen **Grundlagen** für die Entwicklung und Nutzung informationsverarbeitender Prozesse
  - Beispiele für Algorithmen im täglichen Gebrauch sind Spielregeln, Gebrauchsanweisungen, Bastelanleitungen, Kochrezepte, Gewinnstrategien, mathematische Lösungsverfahren, …
  - Ein Algorithmus kann in sehr unterschiedlicher Weise beschrieben werden (unterschiedliche **Syntax**), aber sein Inhalt, d.h. sein Input-Outputverhalten bleibt dasselbe (dieselbe **Semantik**)

Algorithmen

- Einfache Vorgänge automatisieren: Aufgabe Schritt für Schritt ausführen
  - Anweisungsfolgen
- Komplexe Vorgänge erfordern Entscheidungen bzw. Wiederholungen
  - Entscheidung anhand von Vergleichen
  - Wiederholungen durch Schleifen
    - gleichartigen Vorgang wiederholen, solange es nötig ist
    - Abbruchkriterium: Vergleichsoperation
- Algorithmus enthält genauen Ablauf eines Vorgangs, zusammengebaut aus Kombinationen dieser Elementarkonstrukte
  - Algorithmus ist **eindeutig** (eindeutiges Ergebnis)
  - Algorithmus **terminiert** (wird irgendwann sicher fertig)

Algorithmen

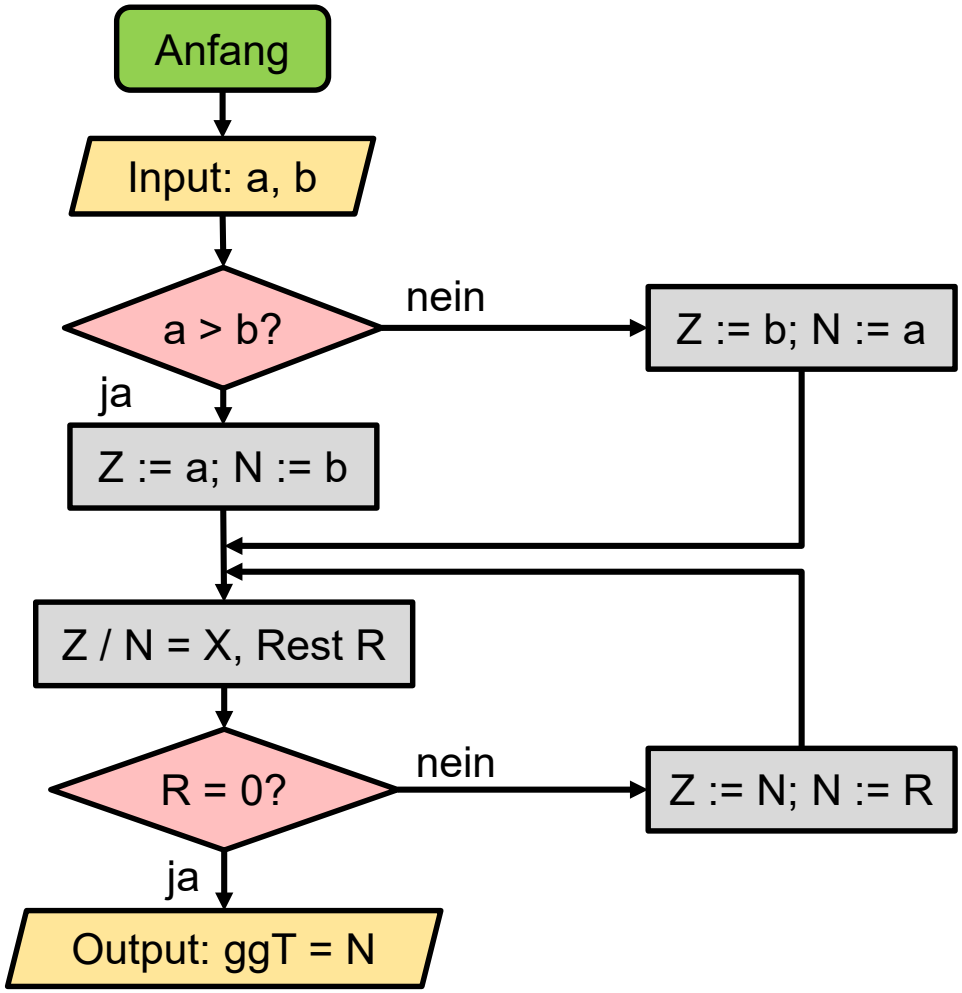
- Beispiel: **Euklidischer Algorithmus** zur Bestimmung des größten gemeinsamen Teilers (ggT) zweier natürlicher Zahlen a, b
- Beschreibung mittels Umgangssprache:
  - Man teile die größere der beiden Zahlen (Input) durch die kleinere
  - Ist diese Division ohne Rest, ist der Nenner der ggT
  - Geht die Division nicht auf, bleibt ein Rest, der nun der neue Nenner wird, und der vorige Nenner wird zum Zähler.
  - Man setze das Verfahren wie am Beginn fort; nach endlich vielen Schritten erhält man einen Rest 0.
    Der zuletzt ermittelte Nenner ist der ggT (Output).
  - Wenn der ggT=1 ist, sind die beiden Zahlen teilerfremd.

Algorithmen

- Beispiel: **Euklidischer Algorithmus** zur Bestimmung des größten gemeinsamen Teilers (ggT) zweier natürlicher Zahlen a = 578 und b = 391
  - Man ermittle den ggT von 578 und 391:
    - 578 : 391 = 1 Rest 187
    - 391 : 187 = 2 Rest 17
    - 187 : 17 = 11 Rest 0
  - Die Division geht auf, **17** ist der ggT von 578 und 391.

Algorithmen

- Euklidischer Algorithmus als Flussdiagramm



Algorithmen

- Beispiel: Euklidischer Algorithmus als Java-Programmcode (Methode, wird später erklärt)

```
public static int ggT(int a, int b) {
    int Z, N, R;
    if(a > b) {
        Z = a;
        N = b;
    } else {
        Z = b;
        N = a;
    }
    R = Z % N;
    while(R != 0) {
        Z = N;
        N = R;
        R = Z % N;
    }
    return N;
}
```

Programme

- Ein Computer-Programm ist ein streng formalisierter, eindeutiger und detaillierter Algorithmus, der maschinell ausgeführt werden kann
  - daraus folgt: Ein Programm ist die Realisierung eines Algorithmus (nur verfeinerter und konkreter als ein allgemein beschriebener Algorithmus, da dieser nun in einem ganz bestimmten Formalismus – der jeweiligen Programmiersprache – notiert ist)

Programme

- Programme sind Anweisungen, die einen Rechner veranlassen, bestimmte Dinge nach dem EVA-Prinzip auszuführen
  - EVA: Eingabe - Verarbeitung - Ausgabe
  - Zur Speicherung der Anweisungsfolgen im Rechner oder auf einem externen Medium benutzt man eine vorher festgelegte Codierung, die jedem Befehl eine bestimmte Bitfolge zuordnet
  - Bei der Erstellung werden Programme als ASCII- oder Unicode-Text formuliert (**Quelltext**)
  - Ein Compiler übersetzt diesen Text in eine Reihe von Befehlen, die sogenannte **Maschinensprache**, die der Rechner „versteht"

Programme

- **Wiederholung aus dem Kapitel "Scripting"**
- Kompilation versus Interpretation
  - Der erste Schritt beim Programmieren besteht immer darin, den Quelltext des Programms aufzusetzen
  - dieser ist einfacher, unformatierter ASCII-Text, in dem man unter Beachtung der Syntax der verwendeten Programmiersprache die Befehle niederschreibt, die der Rechner bei Ausführung des Programms abarbeiten soll
  - der Rechner kann allerdings den Quelltext nicht direkt abarbeiten; dazu ist ein zweiter Schritt erforderlich, nämlich die Umwandlung des Quelltextes in Maschinencode (Binärcode als Folge von ‘0‘ und ‘1‘ ; stellen einen kleinen Satz elementarer Befehle dar, die der Prozessor des Rechners ‘versteht‘)
  - Zur Umwandlung eines Quelltextes in Maschinencode gibt es zwei unterschiedliche Konzepte: **Kompilation** und **Interpretation**

Programme

- Kompilation
  - Hierzu wird ein spezielles Programm (der Compiler) aufgerufen, das den Quelltext des Programms in Maschinencode übersetzt
  - das Ergebnis ist eine von der CPU ausführbare Datei, die als eigenständiges Programm direkt aufgerufen und ausgeführt werden kann
  - Kompilierte Programme sind meist schneller und effizienter als vergleichbare interpretierte Programme
  - Kompilierte Programme sind nicht portabel, da der vom Compiler erzeugte Maschinencode immer auf einen bestimmten Prozessortyp abgestimmt ist und von anderen Prozessoren nicht verarbeitet werden kann
  - Beispiele für kompilierte Programmiersprachen sind Pascal, C/C++ und Java

Programme

- Interpretation
  - Hier fallen Übersetzung und Ausführung des Programms zusammen, das macht der Interpreter
  - Dieser liest das Programm Zeile für Zeile ein, erzeugt entsprechenden Maschinencode und lässt diesen direkt ausführen
  - Interpretierte Programme sind in der Ausführung langsamer als vergleichbare kompilierte Programme, da im Hintergrund immer der Interpreter läuft und die Zeit für die Umsetzung in Maschinencode hinzugerechnet werden muss
  - Interpretierte Programme können auf andere Rechner portiert werden, da sie nur als Quelltext vorliegen (vorausgesetzt ein passender Interpreter ist auf diesen installiert)
  - Beispiele für interpretierte Programmiersprachen sind Basic, Perl, JavaScript und PHP

Java

- Java ist laut einem Whitepaper aus dem Jahr 1995 von JAMES GOSSLING, einem der Erfinder von Java, eine **einfache**, **objektorientierte**, **verteilte**, **interpretierte** (!), **robuste**, **architekturneutrale**, **portable**, **leistungsfähige**, **parallelisierbare** **dynamische** Programmiersprache.
  - Beschreibung enthält einige wichtige Grundbegriffe der Informatik

Java

- **Einfachheit**
  - Java ist an die weit verbreiteten Sprachen C und C++ angelehnt, vermeidet aber komplizierte Konstrukte dieser Sprachen, hat einen reduzierten Sprachumfang, erleichtert dem Programmierer die Arbeit, indem bestimmte Dinge automatisiert werden, um die sich der Programmierer nicht mehr kümmern muss.
  - Java beinhaltet viele **Bibliotheken** für unterschiedlichste Zwecke, die **vordefinierte Programmteile** enthalten, die vom Programmierer genutzt werden können und das Programmieren erleichtern. Bibliotheken werden dazu **importiert**, d.h. in das eigene Programm eingebunden bzw. mit diesem verknüpft.

Java

Objektorientierung

- neueres Programmierparadigma:
  OOP – objektorientierte Programmierung
- Modellierung der Wirklichkeit als **Objektstruktur**
- Definieren von **Klassen**
- **Objekte** sind **Instanzen** von Klassen
- **Algorithmen** einer Klasse sind in **Methoden** definiert
- **Attribute** legen die **Eigenschaften** des Objekts fest
- Klasse **kapselt** Daten in Attributen und zugehörigen Algorithmen in Methoden
- **Vererbung** ermöglicht die Ableitung einer spezielleren Klasse aus einer allgemeinen Klasse, dabei werden die ursprünglichen Methoden und Attribute der **Basisklasse** (Elternklasse, Oberklasse) an die **abgeleitete Klasse** vererbt, können verändert und durch zusätzliche Attribute und Methoden ergänzt werden

Java

- **verteilt**
  - Java unterstützt die im **Internet** verbreiteten **Protokolle**, beispielsweise TCP/IP
  - Zugriff auf Objekte über das Netz mittels URLs
  - Im Web kann Java **client-seitig** oder **server-seitig** genutzt werden
    - **Java Applets**
      werden durch den Client vom Server geladen und auf dem Client-Rechner ausgeführt
    - **Java Server Pages** (JSP)
      basierend auf JHTML (JavaHTML) können HTML und Java-Quellcode auf einem Server innerhalb einer Seite genutzt werden, der JSP-Server führt bei Abruf der Seite den Java-Code (Servlet) aus und liefert Ausgaben des Programmes zusammen mit den HTML-Bestandteilen an den Client aus.
    - **JavaServer Faces** (JSF)
      Framework-Standard für Programmierung von Webapplikationen

Java

- **interpretiert**
  - wir haben gehört: Java-Quellcode wird **kompiliert**
  - dabei entsteht aber kein Maschinencode, sondern **Java Bytecode**
  - der Bytecode wird von der **Java Virtual Machine** (JVM) **interpretiert**
  - JVM kann den Bytecode **optimieren** und auch in Maschinencode übersetzen (kompilieren)

Java

- **Robustheit**
  - **stark typisierte** Sprache, **explizit** typisiert
    - **Variablen** haben einen vorab festgelegten **Typ**, der bestimmt, welche Werte die Variable annehmen kann, z.B. nur ein einzelnes ASCII-Zeichen, eine ganze Zahl in einem bestimmten Bereich, eine Gleitkommazahl mit einem festgelegten Wertebereich
    - **explizite** oder **automatische Typumwandlung**, z.B. bei der Multiplikation einer Ganzzahl mit einer Gleitkommazahl
  - Überprüfungen auf mögliche Probleme beim Übersetzen (Compiler)
  - Überprüfungen zur Laufzeit, wenn das übersetzte Programm ausgeführt wird

Java

- **Sicherheit**
  - **Security Manager** und **Class Loader** erlauben nur den Zugriff auf bestimmte, festgelegte Objekte
  - wichtig beispielsweise bei Applets, Sandbox-Konzept
  - Unterstützung für Kryptographie, z.B. signierte Applets

Java

- **Architekturneutralität**
  - Java Bytecode ist unabhängig von der konkreten Rechnerarchitektur
  - **JVM abstrahiert** von der konkreten Hardware
  - Bytecode, der beim Kompilieren des Quelltextes entsteht, ist nicht spezifisch für eine bestimmte Maschine, kann weitergegeben und auf beliebigen Systemen ausgeführt werden, für die eine Java Virtual Machine existiert
  - sehr praktisch für verteilte Ausführung, z.B. client-seitige Applets

Java

- **Portabilität**
  - gleiches Programm kann auf unterschiedlichen Rechnern ausgeführt werden und zeigt gleiches Verhalten
    - Wertebereich von Variablen unabhängig von Prozessor (z.B. Wortbreite 32 Bit oder 64 Bit)
    - Bibliotheken für grafische Benutzeroberflächen (Fenster, Knöpfe, Textfelder, …) unabhängig vom Betriebssystem
    - maschinenunabhängiger Bytecode

Java

- **Leistungsfähigkeit**
  - Kompilierter Code ist schneller als interpretierter, da Quelltext nur einmal eingelesen und anhand der Grammatik der Sprache **geparst** werden muss
  - entstehender Bytecode konzipiert für schnelle Ausführbarkeit
  - Ausführung von Bytecode auf der JVM erlaubt **Optimierung** bei der Ausführung
  - durch Bibliotheken und Automatisierung bestimmter Aufgaben jedoch Minderung der Leistungsfähigkeit (**Performance**) möglich
  - Da Bibliotheken universell einsetzbar sein sollen, sind sie nicht für einen bestimmten Zweck programmiert, sondern sollten möglichst für viele Zwecke nutzbar sein, daher Programmcode enthalten, der in bestimmten Fällen unnötig ist, aber trotzdem ausgeführt werden muss (**Overhead**)

Java

- **Parallelisierbarkeit**
  - Java enthält Sprachkonstrukte, die einen parallelen (gleichzeitigen) Ablauf von eigenständigen Programmteilen ermöglichen (**Multithreading**)
  - **Thread**: "Ablauffaden"
  - Moderne CPUs enthalten **mehrere Rechenkerne**, die gleichzeitig (**parallel**) unterschiedlichen Programmcode abarbeiten können
  - Gesamtaufgabe kann teils durch gleichzeitige Abarbeitung von Teilen beschleunigt werden
  - in der Praxis einige interessante Fragestellungen der Informatik: **Synchronisation**, **gegenseitiger Ausschluss**, **Deadlocks**, …

Java

- **Dynamik**
  - Bibliotheken werden erst bei Ausführung des Programms geladen (zur Laufzeit) und können somit angepasst und geändert werden
  - Interfaces legen die Kommunikation zwischen Software-Modulen fest. Es ist also nur festgelegt, wie die Module Daten austauschen. Die eigentlichen Implementierungen können noch während der Ausführung dynamisch geändert werden.

Java

- Programmierung
  - Java-Umgebung für unterschiedliche Betriebssysteme von Oracle verfügbar
    - Java Development Kit (JDK) enthält Compiler, ermöglicht es, eigene Programme zu erstellen
    - Java Runtime Environment (JRE) erlaubt die Ausführung von Java Bytecode mittels der JVM, kein Compiler enthalten
  - Open-Source-Implementierung OpenJDK
  - aktuelle Version: Java SE Platform 11

Java

- Programmierung
  - Quelltext erstellen
    - mit Texteditor
      z.B. Emacs, vi, Notepad++, …
    - mit integrierter Entwicklungsumgebung (IDE)
      z.B. Eclipse, NetBeans, IntelliJ IDEA, BlueJ, …
    - als menschenlesbarer ASCII- oder Unicode-Text (UTF-8)
    - Java erfordert Definition einer Klasse (siehe OOP), in der das Programm implementiert wird
    - speichern mit `.java als` Dateianhang
    - Name der Datei muss dem Namen der definierten Klasse entsprechen,
      z.B. Klasse `HelloWorld` in Datei `HelloWorld.java`
    - beim Aufrufen des Programms wird die `main()`-Methode dieser Klasse ausgeführt
    - diesen Rahmen müssen wir zunächst als nötig und vorgegeben ansehen, weiter Erklärungen folgen später

Java

- Programmierung
  - Quelltext

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

  - in Datei `HelloWorld.java` speichern

Definition der Klasse

`main()`-Methode wird bei Programmstart aufgerufen

Geschweifte Klammern zur Bildung von Blöcken, vgl. öffnende und schließende Tags in HTML

Java

- Kompilieren der Quelltextes
  - durch Funktion der genutzten IDE
  - durch Aufruf des Java-Compilers auf der Kommandozeile
    - Voraussetzung:
      wir befinden uns im gleichen Verzeichnis wie der Quelltext
    - **`javac`** mit Argument *`Klassenname`*`.java` aufrufen,
      z.B. `javac HelloWorld.java`
    - erzeugt Java Bytecode in Datei *`Klassennname`*`.class`,
      z.B. `HelloWorld.class`

Java

- Ausführen des Bytecodes durch die Java Virtual Machine
  - wieder mittels IDE möglich
  - durch Aufruf des Bytecode-Interpreters (JVM) auf der Kommandozeile
    - Voraussetzung:
      wir befinden uns im gleichen Verzeichnis wie die Bytecode-Datei
    - **`java`** mit Argument *`Klassenname`* aufrufen,
      z.B. `java HelloWorld`
    - die Dateierweiterung `.class` muss weggelassen werden
    - JVM führt die `main()`-Methode der Klasse aus

Java

- kleiner Ausflug: Android
  - Entwicklungsumgebung für Android Apps basiert auf Java, aktuell auf OpenJDK
    - zusätzliches Android-SDK
    - früher (bis Android 4.x): eigene virtuelle Maschine *Dalvik,* optimiert auf aktuelle Prozessoren
    - Apps werden in Java programmiert
    - Kompilierung zu Java Bytecode mit üblichem Java-Compiler
    - der entstehende Bytecode wird dann mit **Cross-Assembler** `dx` in Bytecode für die Dalvik-VM übersetzt (dex)
    - fertige Anwendung in `.apk`-Paket (Android Packet) verpackt
    - ab Android 5.0 wird Anwendung bei der Installation in nativen Bytecode für den jeweiligen Prozessor umgewandelt
      - Ausführung in ART (Android Runtime)
    - Entwicklungsumgebung Android Studio mit IDE (IntelliJ) kostenlos
    - Patent-/Lizenzstreitigkeiten zwischen Google und Oracle