



# Java Einführung

## **SOFTWARE-ENTWICKLUNG**

Kapitel 1

# Inhalt dieser Einheit



- Phasen eines Softwareprojekts
- Algorithmus
- Überblick über (Programmier-) Sprachen
- Wie funktioniert JAVA?

# Software Engineering

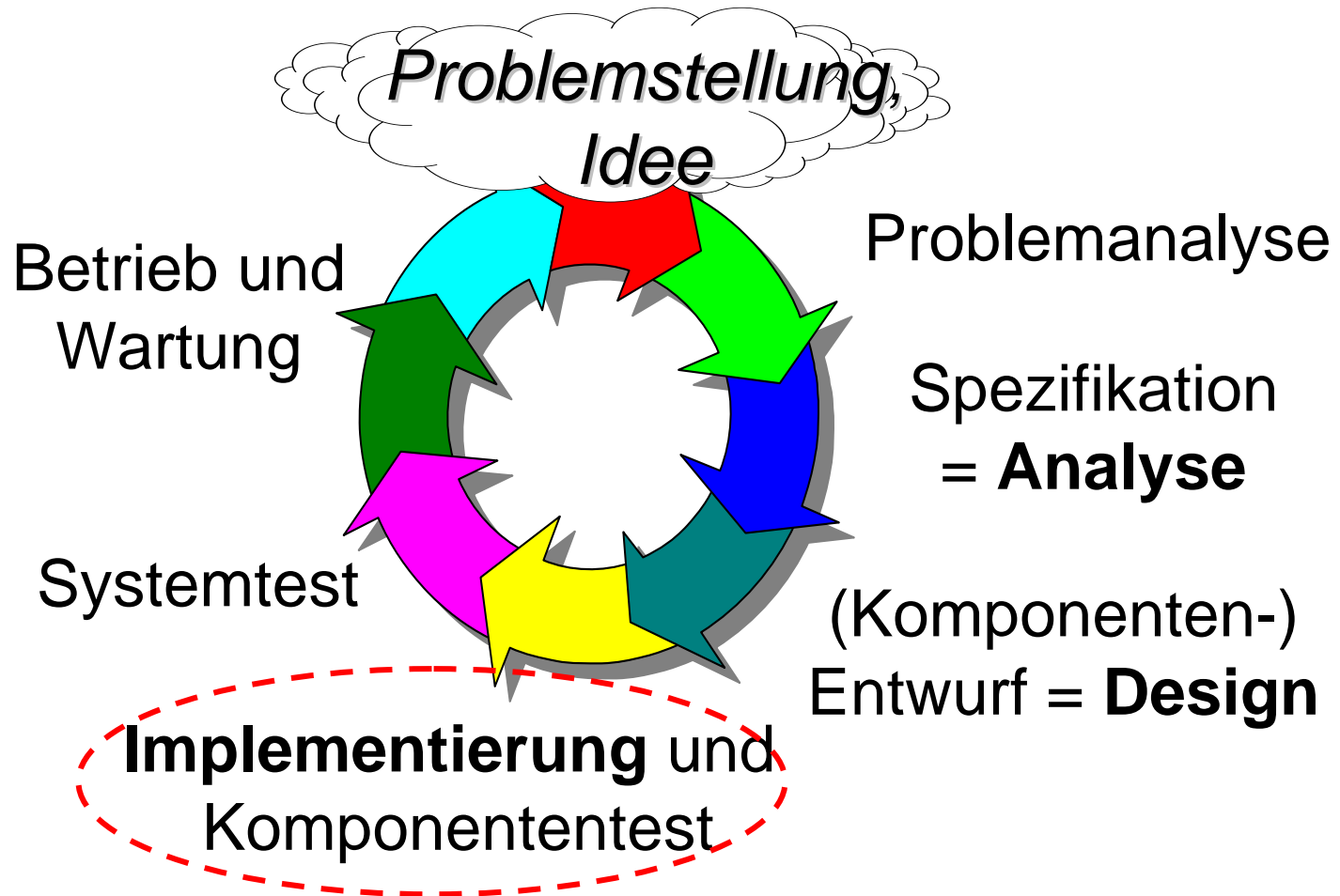
Fachgebiet der Informatik stellt

- Methoden
- Werkzeuge

zur Herstellung und Anwendung von Software bereit.

# Die Phasen eines Softwareprojekts

(klassisch sequentielles Software-Life-Cycle-Modell)



# Problemanalyse und Planung

**Was?**

- Erhebung des Ist-Zustandes
- Abgrenzung des Problembereichs
- Skizzieren der Bestandteile des geplanten Systems
- erste Abschätzung des Umfangs und der Wirtschaftlichkeit
- Erstellung eines groben Projektplans
- Benutzeranalyse
- Aufgabenanalyse

# Analyse: Spezifikation

## Was?

- Genaue Festlegung des Leistungsumfanges des geplanten Systems (Pflichtenheft)
- Umsetzung der Anforderungen in ein **Modell**, welches die künftige Funktionalität des zu entwickelnden Produktes vollständig beschreibt
- Funktionsmodelle, Datenmodelle, Ereignismodelle, Interaktionsmodelle, Workflow-Modelle
- noch unabhängig von der Implementierung

# Design: System- und Komponententwurf

Wie?

- Umsetzung, der in der Spezifikation erstellten Modelle in eine formale Form
- Festlegung, welche Systemkomponenten die vorgegebenen Anforderungen abdecken werden
- Definition der **Systemkomponenten** (Schnittstellen!)
- Entwurf der **algorithmischen** Struktur (Pseudocode)
- Entwurf der **Datenstrukturen**
- unabhängig von der Implementierung!

# Implementierung und Komponententest

- Umsetzung der Ergebnisse der Entwurfsphase in eine rechnerausführbare Form
- Verfeinerung der Komponenten und Algorithmen
- Codierung in der gewählten Programmiersprache
- Komponententests

# Systemtest

- Wechselwirkungen der Systemkomponenten unter realen Bedingungen prüfen
- Fehler aufdecken
- sicherstellen, dass die Systemspezifikation erfüllt ist (aus Pflichtenheft)
- Freigabe der Software

# Betrieb und Wartung

- Fehler, die erst während des tatsächlichen Betriebs auftreten beheben
- Systemänderungen und Systemerweiterungen durchführen

# Andere Software-Engineering Modelle

- Wasserfall-Modell
- Spiral-Modell
- Prototyping-orientiertes Life-Cycle-Modell
- Objektorientiertes Life-Cycle-Modell
- Objekt- und prototyping-orientiertes Life-Cycle-Modell

# Beschreibung von Abläufen

(Bsp. Kochrezept)

*„Man nehme 2 dag Schmalz und erhitze es in einer Pfanne. Inzwischen versprudelt man 3 Eier mit einer Brise Salz. Wenn das Fett heiß ist, gieße man die Eier in die Pfanne. Mit einer Gabel umrühren, bis die Eier stocken, sofort heiß servieren...“*

- Kochrezepte
- Gebrauchsanweisungen
- Algorithmen
- Programme

# Algorithmus (Definition)

Ein **Algorithmus** ist ein

- Plan zur Lösung gleichartiger Probleme mit Hilfe einer **endlichen Folge eindeutiger, ausführbarer Schritte**
- bestehend aus
  - elementaren Anweisungen
  - in einer geeigneten Sprache
  - die sequentiell (schrittweise) durchgeführt werden.

Ein **Programm** ist ein Algorithmus formuliert in einer bestimmten Programmiersprache.

# Algorithmus

( Bsp. Telefonnummersuche)

*"Suche die Telefonnummer zu einem gegebenen Namen in einem beliebigen Adressbuch"*

1. Gehe zum ersten Eintrag im Adressbuch
2. Vergleiche den aktuellen Eintrag mit dem gesuchten Namen
3. Bei Gleichheit: gib die zugehörige Telefonnummer aus und halte an sonst gehe zu Schritt 4
4. (Ungleichheit) Wenn noch Einträge im Adressbuch vorhanden sind: nimm' den nächsten Eintrag und gehe zu (2) sonst gehe zu Schritt 5
5. melde Misserfolg und halte an

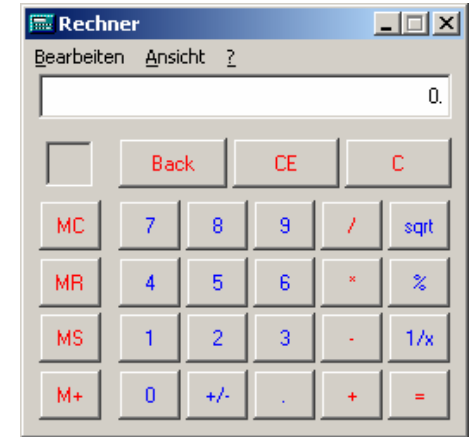
Name	Tel.Nr.
Neumann, Gustaf	12235
Neumann, Gustav	13356

# Definition einer Sprache

Wie definiert sich eine Sprache?

- Aus der Menge der **Symbole** (Worte) können Sätze gebildet werden.
- **Syntax**: Die Regeln für das Bilden **gültiger Sätze** aus diesen Symbolen ("Grammatikregeln").
- **Semantik**: Die Bedeutung der gültigen Sätze.
- Bsp:  $2+4=7$  ist in der Sprache der Mathematik syntaktisch richtig, aber semantisch falsch.

# Bsp: Regelgrammatik Syntaxregeln (BNF)



$\langle \text{Ziffer} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

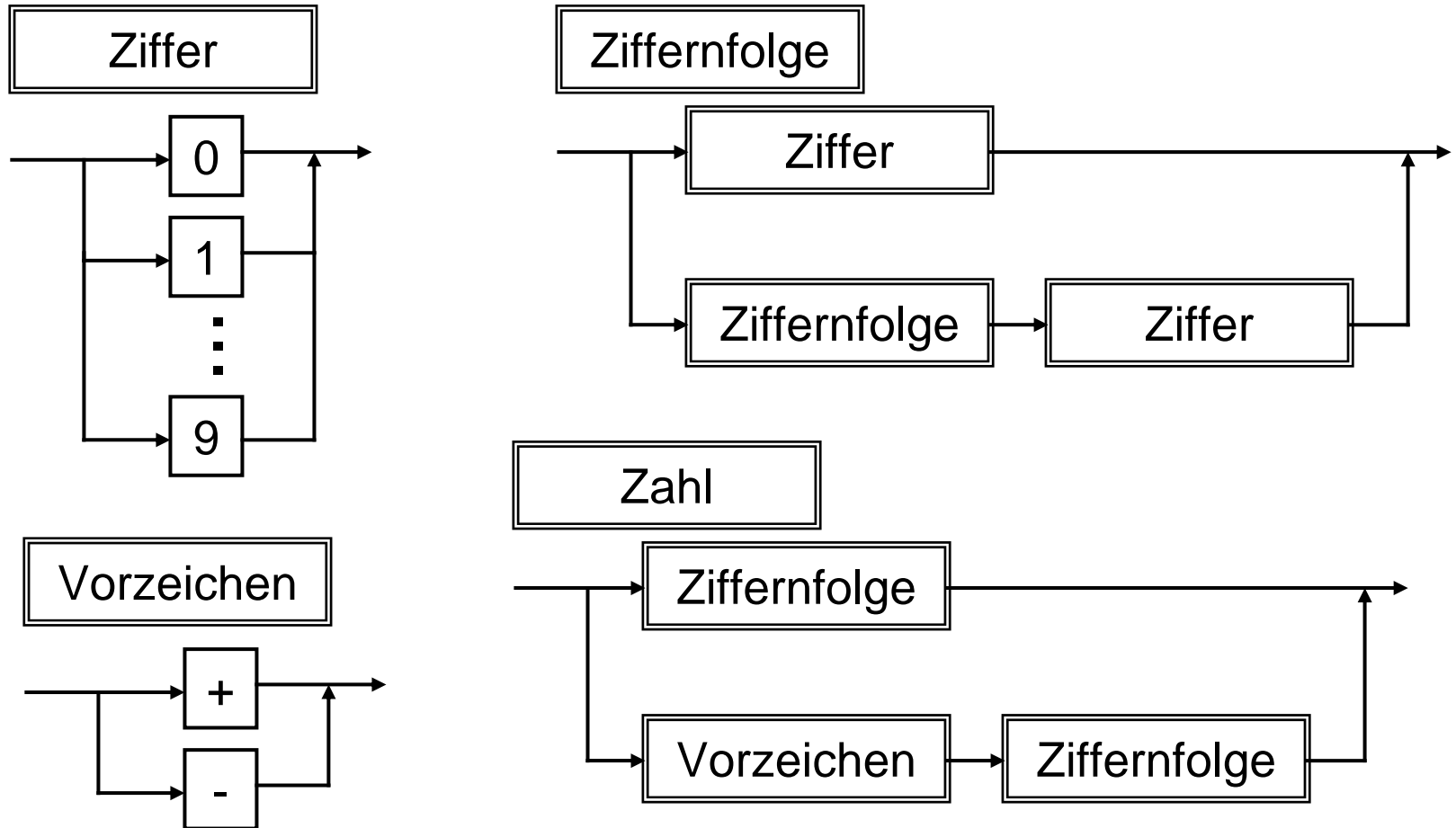
$\langle \text{Vorzeichen} \rangle ::= + \mid -$

$\langle \text{Ziffernfolge} \rangle ::= \langle \text{Ziffer} \rangle \mid \langle \text{Ziffernfolge} \rangle \langle \text{Ziffer} \rangle$

$\langle \text{Zahl} \rangle ::= \langle \text{Ziffernfolge} \rangle \mid \langle \text{Vorzeichen} \rangle \langle \text{Ziffernfolge} \rangle$

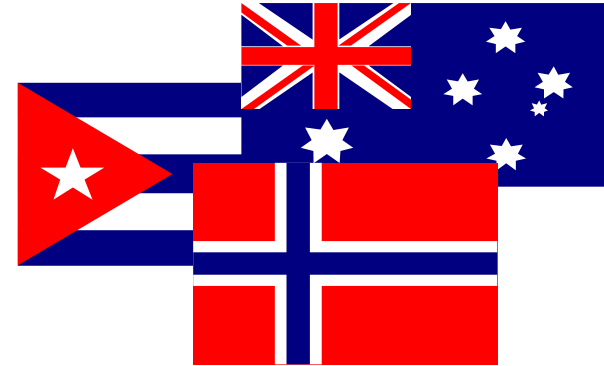
- **Terminale Symbole** werden nicht mehr abgeleitet  
{0,1,2,...,9}
- **Nicht-terminale Symbole** gekennzeichnet durch  $\langle \dots \rangle$
- $\mid$  bedeutet eine Auswahl (oder)
- Alles was von  $\langle \text{Zahl} \rangle$  (Startsymbol) abgeleitet werden kann, ist eine gültige Zahl

# Bsp: Syntaxdiagramm



# Natürliche versus formale Sprache

- Natürliche Sprachen sind vielfach syntaktisch und/oder semantisch mehrdeutig.  
(„Dings“ wird aus dem Kontext erkannt)
- Natürliche Sprachen sind fehlertolerant.
- Der vorgegebene Wortumfang natürlicher Sprachen ist sehr groß.  
(80000 Worte Langenscheidt vs. 51 Java)
- Eine Programmiersprache ist eindeutig und ist primitiv genug, um in Maschinencode übersetzt werden zu können.



# Java-Wortschatz

## reservierte Worte

abstract	boolean	break	byte	case
catch	char	class	continue	default
do	double	else	extends	false
final	finally	float	for	if
implements	import	instanceof	int	
long	native	new	null	
private	protected	public	return	
static	super	switch	synchronized	
this	throw	throws	transient	
try	void	volatile	while	
true	short	interface	package	assert
enum	strictfp			

\* derzeit ohne Verwendung

byvalue*	cast*	const*	future*	generic*
goto*	inner*	operator*	outer*	rest*
var*				

# Programmiersprachen Definition

## Eine Programmiersprache

- ist eine Sprache zur Formulierung von Rechenvorschriften bzw. Anweisungen (Datenstrukturen und Algorithmen), die von einem Computer ausgeführt werden können,
- hat eine eindeutig definierte
  - **Syntax** (welche Zeichenfolgen sind erlaubt) und
  - **Semantik** (was bewirken die Zeichenfolgen auf dem Rechner),
- bildet die wichtigste Schnittstelle zwischen Rechner und Programmierer.

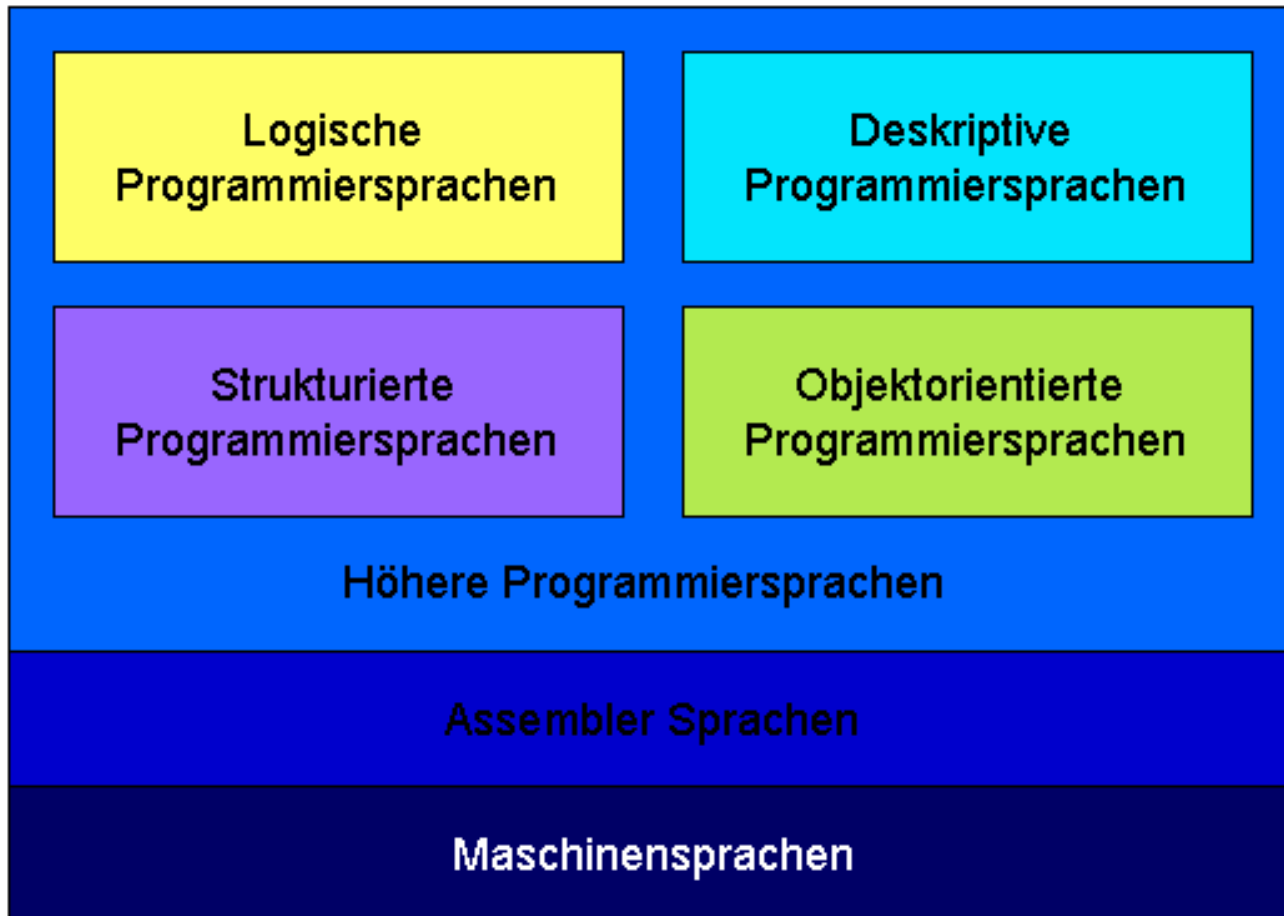
# Wie kann ich eine Sprache lernen?

## ÜBEN und ANWENDEN!

Wie beim Lernen eines Musikinstruments!  
Notenkenntnis und  
Erkennen der Tasten  
ist zuwenig!



# Programmiersprachen



# Programmiersprachen

## Übersicht

- **Maschinensprache**

Programme in Maschinensprache bestehen aus Maschinenbefehlen, die als Folge von Nullen und Einsen im Rechner dargestellt werden. Nur Maschinensprache kann vom Prozessor verarbeitet werden.

- **Assembler Sprachen**

Die Bitkombinationen der Maschinenbefehle werden durch ein leicht zu merkendes Symbolwort ausgedrückt werden (MOV A L).

- **Höhere/symbolische Programmiersprachen**

Sind an die menschlichen Denkweisen angepasst (sie sind problemorientiert und nicht mehr maschinenorientiert). Höhere Programmiersprachen erlauben die Formulierung von Programmen in einer vom Zielrechner unabhängigen Notation. Mit Hilfe eines Übersetzungsprogramms (Compiler) werden Programme einer höherer Programmiersprache in Maschinenprogramme übersetzt.

# Höhere Programmiersprachen

## Übersicht

- **Strukturierte Programmiersprachen**  
Höhere Programmiersprachen, die das Modulkonzept unterstützen  
*BASIC, PASCAL, C, Modula*
- **Objektorientierte Programmiersprachen**  
Höhere Programmiersprachen, welche Daten und Anweisungen nicht trennen sondern zusammen als Objekte behandeln. Objekte stehen miteinander in Beziehung und kommunizieren miteinander durch Versenden von Nachrichten (Messages). Das Empfängerobjekt hat für jede Nachricht, die es “verstehet”, einen eingebauten Algorithmus (Methode genannt). *C++, C# und Java*
- **Deskriptive Programmiersprachen** dient zur Beschreibung der Eigenschaften gesuchter Informationen (üblicherweise Daten aus einer Datenbank) *SQL*
- **Logische Programmiersprachen**  
Die logische Programmierung beruht auf der Erkenntnis, dass einer Untermenge der Prädikatenlogik eine prozedurale Interpretation gegeben werden kann. *PROLOG*



# Entwicklung von prozeduralen Programmiersprachen

**John v. Neumann: Programmierbare Rechenmaschine**

1940

**IBM: PL/1**

General Purpose PL

1964

**N. Wirth: Pascal**

Strukturierte Programmierung

**Dahl: Simula**

Simulation

1967

1971

**N. Wirth: Modula-2**

Verbesserte Datenkapselung

**D. Ritchie (AT&T): C**

Systemprogrammierung

1974

1980

**B. Stroustrup (AT&T): C++**

Objektorientierte Erweiterung von C

1987

**Sun: Java**

vereinfachte und gestutzte Weiterentwicklung von C++

1995

# Java-Geschichte

- 1991: Projekt ``Green`` (consumer electronics) bei Sun zunächst basierend auf C++, das sich aber im Hinblick auf Sicherheit, Portabilität und Zuverlässigkeit als unzureichend herausgestellt hatte => Entwicklung einer eigenen Sprache ``Oak``, später Java
- 1994: Entwicklung des HotJava-Browsers (**Java Applets**)
- 1995: offizielle Vorstellung von Java, Netscape lizenziert Java
- 1996: Gründung von JavaSoft bei Sun, Release 1.0 des Java Development Kits (JDK)
- 1997: JDK 1.1
- 1999: JDK 1.2 = Java 2 Platform
- 2000: JDK 1.3
- 2001: SDK 1.4
- 2005: JDK 5.0 = Version 1.5



# Was ist Java?

- **Objektorientiert**
- **Einfach** (keine Pointer, automatische Garbage Collection, nur einfache Vererbung)
- **Plattformunabhängig** (durch Bytecode)
- **Verteilt/Netzwerkfähig** (Internet)
- **Sicher** (Sandbox)
- **Multithreading-fähig** (Parallelverarbeitung)

# Grundlagen: Ausführbare Programme

- Der **Quellcode** beinhaltet Anweisungen, die noch nicht direkt durch ein Computersystem ausgeführt werden können. Für die Ausführung der Anweisungen muss der Quellcode erst in **Maschinencode** übersetzt werden.
- Es gibt zwei Möglichkeiten, Quellcode einer höherer Programmiersprache in Maschinencode zu übersetzen:
  - **Interpreter**: Nach dem Start des Interpreters wird jeweils eine Anweisung des Quellprogramms nach der anderen in Maschinenbefehle übersetzt, geprüft und sofort ausgeführt.
  - **Compiler**: Übersetzt das Quellprogramm vollständig in Maschinencode, der zu einem beliebigen Zeitpunkt vom Betriebssystem geladen und ausgeführt werden kann.

# Ausführen von Java

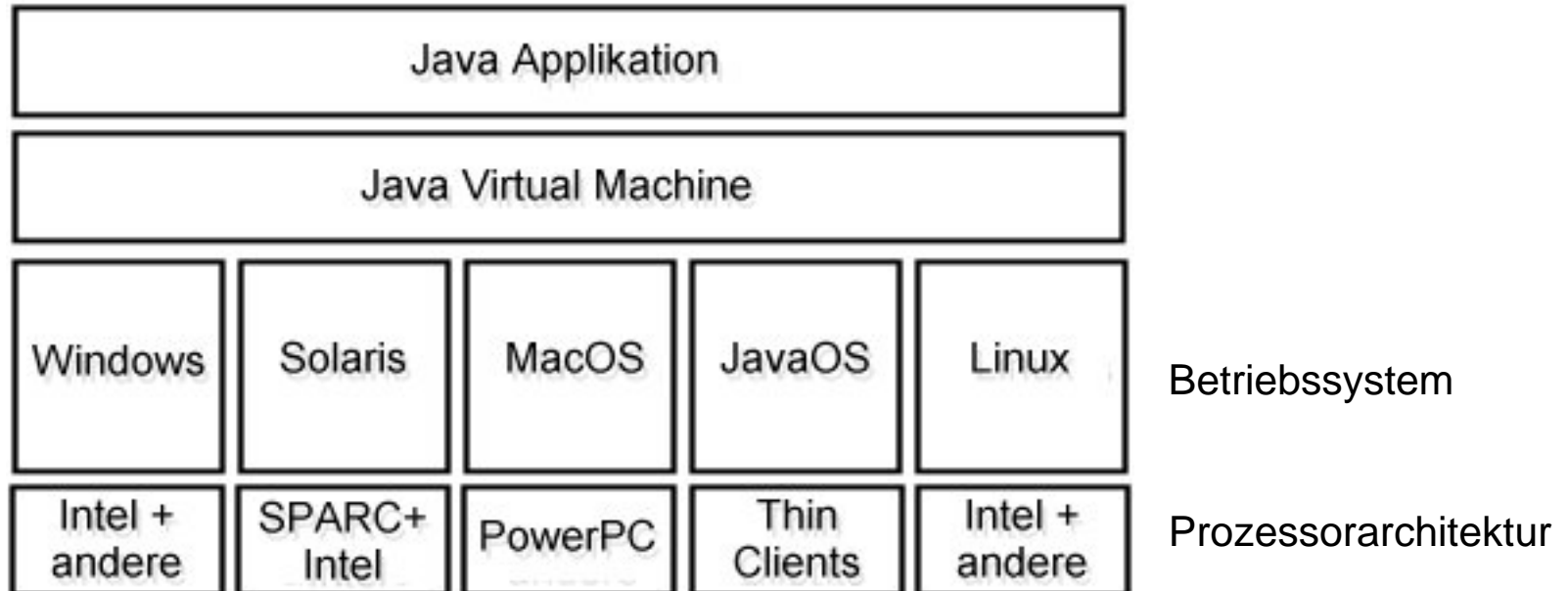
- Java kombiniert Compiler und Interpreter
- Zunächst wird die Quelldatei mit Hilfe des Java Compilers in **Bytecode** übersetzt.



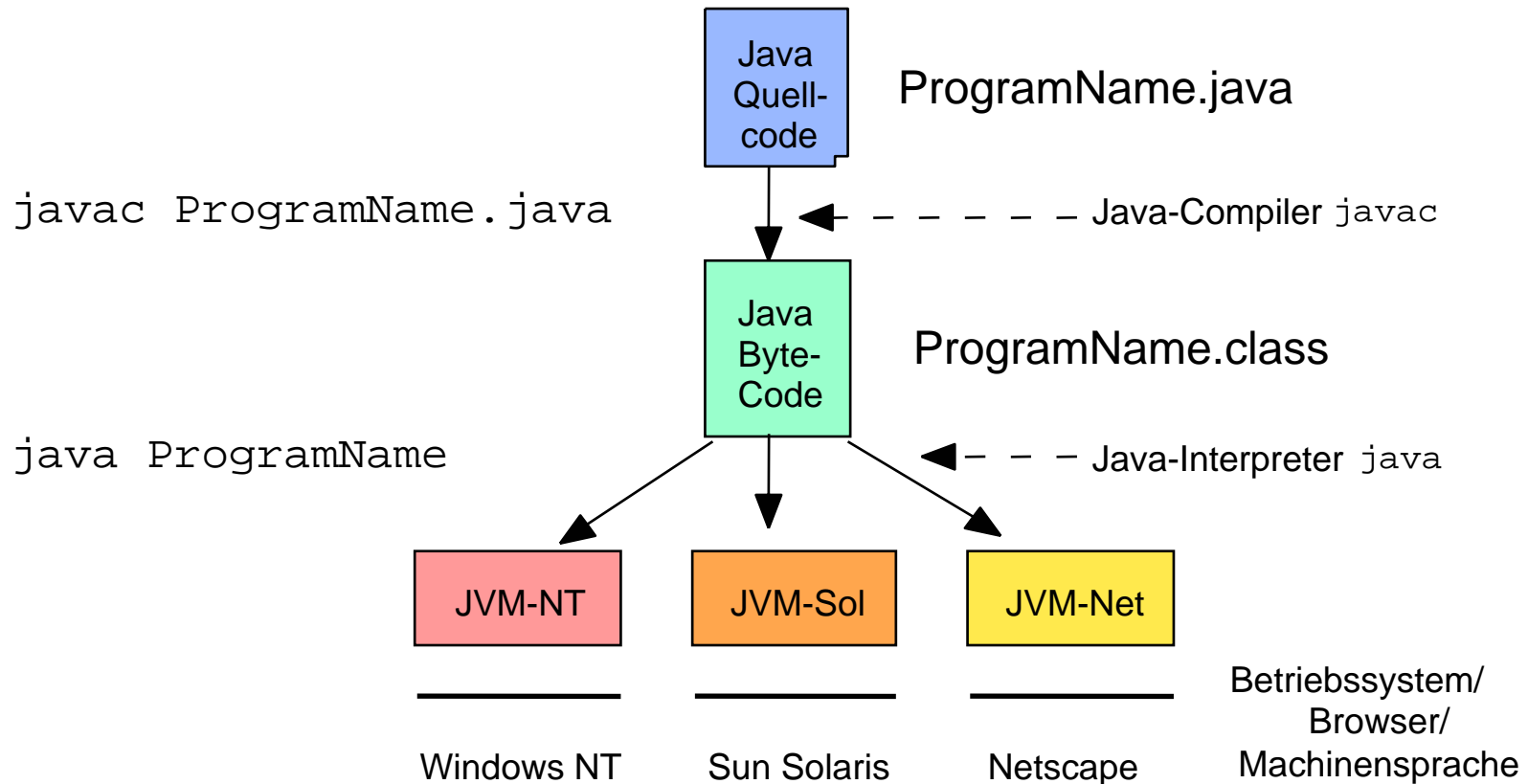
- Wichtig: Die Bytecode-Datei beinhaltet exakt denselben Bytecode, egal von welchem System sie erzeugt wird (Plattformunabhängig!)

# Java Bytecode Interpreter

- Der Bytecode kann noch nicht direkt ausgeführt werden. Dazu wird im zweiten Schritt ein Java **Bytecode Interpreter** eingesetzt.
- Jedes System hat seinen eigenen Java Bytecode Interpreter, die „Java Virtual Machine“ (JVM)



# Java - Übersetzung



# Programmcode

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

## “Quellcode”

Der Programmcode muss erst in **die** für den Computer verständliche Sprache übersetzt werden!

# Java Applikationen vs. Applets vs. Java-Script

- **Java-Applikationen:** Anwendungen auf Betriebssystemebene, z.B. wird unter Windows gestartet und öffnet ein eigenes Fenster.
- **Java-Applets:** Java-Anwendungen, die in einem WWW-Browser ablaufen. Sind in HTML-Seiten eingebettet und werden automatisch über das Internet heruntergeladen; z.B. im Internet Explorer.
- **Java-Script:** Scriptsprache zum Einbetten von einfachen Programmen in HTML-Seiten.

# Selbstkontrolle

- In welchen Phasen läuft ein Software-Projekt ab?
- Welche Klassen von Programmiersprachen gibt es? Und zu welcher Klasse gehört Java?
- Was sind die Eigenschaften von Java?
- Erklären Sie die Schritte vom Java-Quellcode zum ausführbaren Java-Programm.
- Was ist der Unterschied zwischen Java-Applikationen und Java-Applets?

