**Kapitel 2**

**Java**

Eigenschaften

1) vollständig objektoptimiert

- ohne prozedurale Altlasten

2) unkompliziert, einfach und leicht erlernbar

- Syntax ähnlich zu C / C++

- Beschränkung auf das Notwendigste

- keine Pointer

- keine Headerdateien

- keine Präprozessoranweisungen

- keine Mehrfachvererbung

3) plattformunabhängig

- übersetze Java-Programme (Bytecode) sind auf jeder Java-fähigen Plattform

anwendbar

- fest definierter Bereich für Zahlen

4) sicher

- Kein direkter Speicherzugang

- strenge Typüberprüfung

- Keine Programmierung mit Sprachverletzung

5) robust

- keine Rechnerabstürze durch Programmierfehler

- Überprüfung der Speicherzugriffe

- Ausnahmeroutinen zur Fehlerbehandlung

6) multithreaded

- parallele Ausführung von Programmteilen

7) internetfähig

Compiler

Aufgabe: Übersetzt Computerprogramm aus Quellsprache in ein semantisch äquivalentes Programm in einer Zielsprache

Aufbau (2 Phasen)

Analysephase

- lexikalische Analyse zerteilt Code in zusammengehörende Token

- syntaktische Analyse überprüft formale Richtigkeit

- semantische Analyse überprüft semantische Richtigkeit

Synthesephase

- Zwischencodeerzeugung als Grundlage für Optimierung

- Programmoptimierung auf Basis des maschinennahen Zwischencodes

- Codegenerierung in Zielsprache auf Basis von optimiertem Zwischencode

Arten

Nativer Compiler erzeugt Code für die Plattform, auf der er läuft

Cross Compiler erzeugt Code für andere Plattformen

One-Pass Compiler erzeugt Code in einem Durchlauf

Multipass Compiler erzeugt Code in Mehreren Durchläufen

**Linker**

Aufgabe: Fügt einzelne Programmteile zu ausführbarem Programm zusammen

Fügt benötigten Code aus Funktionsbibliotheken zu Hauptcode dazu

Arten

Statisches Linken beim Kompilieren werden benötigte Codings aus Funktionsbibliotheken zum Hauptprogramm hinzugefügt

Dynamisches Linken zur Laufzeit werden Codings aus den Funktionsbibliotheken dazugelinkt

(Bsp. DLL bei Windows) (findet auch bei Java statt)

**Interpreter**

Aufgabe: Übersetzt Programm nicht in eine direkt ausführbare Datei, sondern liest, analysiert und führt ihn aus während der Laufzeit

Vor-/Nachteile

- Auf jeder Rechnerarchitektur möglich

- deutlich langsamer als kompilierte Programme

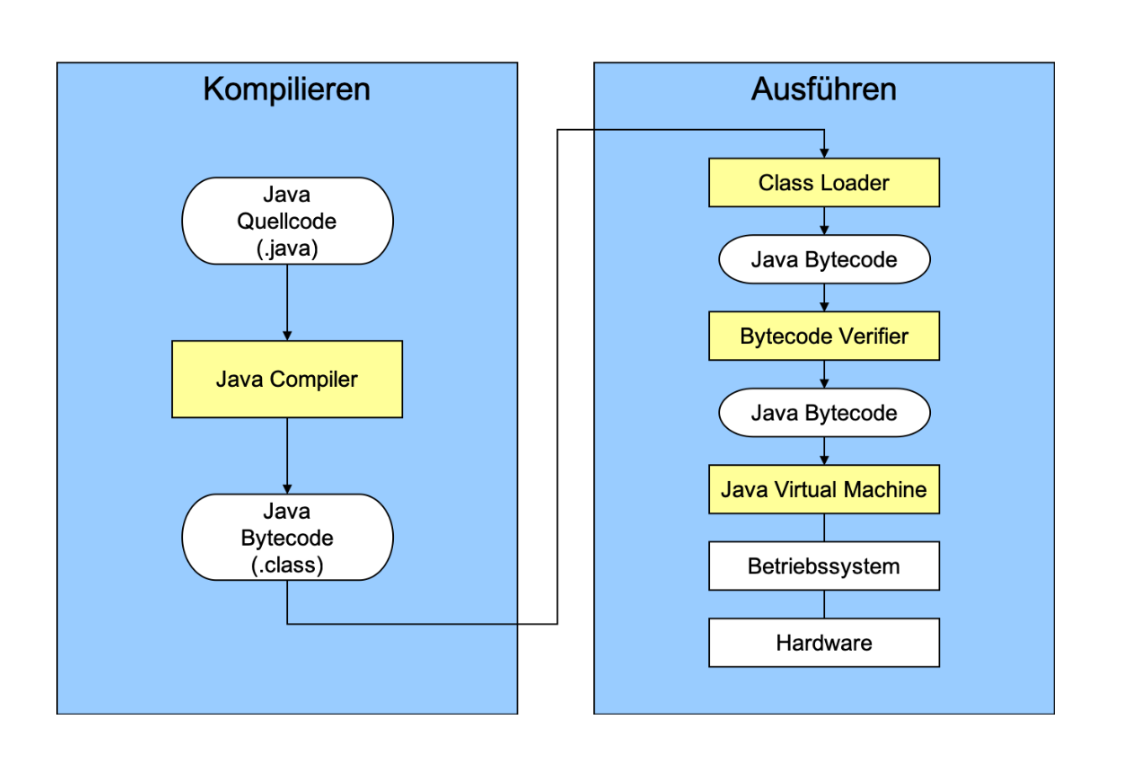
Möglichkeiten zur Geschwindigkeitsoptimierung

- Just-In-Time Compiler Zur Laufzeit wird Quellcode in Maschinencode übersetzt, dieser direkt vom Prozessor ausgeführt. Mehrfach durchlaufene Programmteile müssen nur einmal übersetzt werden. Nicht auf allen Rechnerarchitekturen möglich.

- Bytecode-Interpreter Code wird während der Laufzeit in Bytecode übersetzt, dieser von VM (Interpreter) ausgeführt. Code kann so auf verschiedenen Plattformen ausgeführt werden.

Funktionsweise bei Java

Code wird im Editor erstellt, dann von einem Bytecode-Compiler in Bytecode übersetzt und dann auf einer Java-VM als Interpreter durchgeführt.



Wesentliche Java-Tools

Java-Compiler (javac.exe)

Java-VM (java.exe)

Javadoc erstellt automatische Dokumentation

Class-Loader lädt Java-Klassen in den Arbeitsspeicher und bereiter Ausführung von Java Applikationen vor

Bytecode-Verifier überprüft semantische Korrektheit

überprüft Klassenhierarchie

überprüft jede Methode auf strukturelle Gültigkeit

Datenflussanalyse um z.B. Typfehler zu vermeiden

Das Paketkonzept in Java

- Möglichkeit zur sinnvollen Strukturierung der Klassen

- Pakete sind Ansammlungen von Klassen

- die Klassen verfolgen einen gemeinsamen Zweck

- Jede Klasse ist genau einem Paket zugeordnet

- Paketnahmen können aus mehreren Teilen bestehen und hierarchisch angeordnet sein

- Eine Klasse kann eindeutig über ihren Namen und ihr Paket identifiziert werden