# Grundlagen sowie Vor- und Nachteile Java

Einstieg über Deshmukh (Seite 12-15 (Features/Perspektives) + 16-18 (What Java is not):

**Features (Vorteile) von Java, aus Sicht von**

**Management:**

* plattformunabhängig – WORA (Write Once Run Anywhere)
* hoch performant – gleichauf oder besser als C-Sprachen dank JVM-Verbesserungen
* Sicher – JVM als Sandbox, Security Manager bewacht Zugriffe auf Ressourcen
* vertraut – schnell erlernbar dank der vielen Parallelen zu den C-Sprachen.
* einfacher – durch die wegfallenden Pointer und den Garbage Collector
* mehrere Einspielungs- und Einsatzmöglichkeiten (Client, Server, Browser/Web)
* Java Ökosystem – Bibliotheken: Man muss das Rad nicht immer wieder neu erfinden.
* Abwärtskompatibilität - Alter Code geht durch neue Java/JVM-Versionen nicht kaputt.

**Technologie:**

* (Vor-)kompiliert - Bytecode/class file auf jeder Plattform ausführbar (mit JRE/JVM)
* Alles aus einem Guss (variety of technological solutions under one umbrella)  
  Desktop-Anwendungen über GIU-APIs, sowie Server- und Netzwerk-APIs, uvm.
* Multithreading - Multitasking durch ein Programm möglich (mehrere Threads)
* Verteilbarkeit (Distributed) – Programmaufteilung zwischen Server und Clients
* Automatisches Aufräumen (Garbage Collection) (z.B. verwaiste Objekte)

**Programmierung:**

* objekt-orientiert (Kapselung, Abstraktion, Polymorphismus, Vererbung, Klassen, Obj)
* strukturiert – Modularität zugunsten der Wartbarkeit und Wiederverwendbarkeit
* statisch typisiert (statically typed) – Lege Datentypen vor der Laufzeit fest.
* streng typisiert (strongly typed) – Datentypen während der Laufzeit nicht änderbar.
* Automatische Speicherverwaltung (keine Pointer oder manuelle Speicherzugriffe)
* Praktische Ausnahmen-Behandlung (Exception-Handling, try-catch, Ressourcen)
* Vorgefertigte Java Bibliotheken (Java API) (z.B. für Datei- oder Datenbankoperationen)

**Nachteile (oder was Java nicht kann)**

* Keine Direktausgabe ausführbarer Programmdateien:   
  Dies stimmt nur bedingt. Es lassen sich durchaus Wrapper-EXEs erzeugen, die die JRE einbetten, die JVM starten oder auch durch eine Batch-file (sog. Switch).  
  Grundsätzlich aber werden beim Kompilieren nur class- oder jar-files erzeugt.
* Keine Code-Sicherheit:   
  Class-files können dekompiliert, kompromittiert und als gehackte Version in ein Programm (z.B. in eine jar-File) wieder eingesetzt werden. Man kann bedingt Abhilfe schaffen, indem man Tools nutzt, die den Code unleserlich umschreiben.
* Klobige Nutzeroberflächen:

Die GUI-API von Java stellt nicht die gleichen vertrauten Oberflächen bereit wie die jeweiligen Bibliotheken des Zielsystems. Sie weichen je nach System ab, was dem WORA-Prinzip widerspricht (🡪 write once, test anywhere). Deshalb werden serverseitige Entwicklungsszenarien für Java bevorzugt, statt für Rich/Fat-Clients.

* Nicht (direkt) für Handys (JDK/JRE):

Da es für mobile Plattformen keine JDK/JRE/JVM gibt, können mit dieser entwickelte Java-Programme nicht auf Plattformen ohne diese ausgeführt werden. Indirekt ist Java also an (nur unterstütze) Plattformen gebunden.

Das heißt aber nicht, dass Java grundsätzlich nicht auf Handys läuft, aber eben auf Grundlage anderer Entwicklertools und Laufzeitpaketen (wie Android Studio).

* Nicht so steile Lernkurve:

Simpler als C/C++, aber komplexer als Python. Je komplexer, desto potenter, nehme ich an. Spätestens bei kommerziellen Anwendungen stößt Python an Grenzen. Java ist der Mittelweg (so komplex wie nötig, so wenig wie möglich).

* Mängel der Sprache (subjektiv):   
  Java stand lange in der Kritik für offensichtliche Mängel gegenüber anderen Sprachen, doch dies ist sehr subjektiv, fallabhängig und stets in Verbesserung.
* Nicht für alles geeignet:   
  Jede Programmiersprache hat ihre Grenzen, wie auch Java. Hardwarenahe Programmierung wie Gerätetreiber sind damit nicht möglich, da die JVM eine Sandbox ist, der Security Manager dazwischen sitzt und keine direkten Speicherzugriffe vorgesehen sind. Dies ist aber auch so gewollt. In erster Linie ist Java eine problemorientierte Sprache (lt. Jens kreiert für Wirtschaftsinformatiker).

# Einsatzmöglichkeiten - Wo ist es überall?

Einstieg über Deshmukh (Seite 15-16)

**Desktop Applications**

Sowohl GUI-, also auch CLI-basierte Desktop Anwendungen vom Adressbuch bis hin zur ERP (Enterprise Ressource Management (z.B. SAP ist ein ERP-Programm)) können mit Java entwickelt werden, auch Backgroundservices wie Screen Capturing sind möglich.

**Distributed Applications**

Das Flugticketsystem einer Airline kann serverseitig die Tickets verwalten und dem Kunden per Clientanwendung mit GUI Informationen zur Verfügung stellen (Java RMI).

**Web Applications**

Während seitens des Front-Ends Java Server Pages (JSP) durch JavaScript abgelöst wurde, wird serverseitig (Backend) immer noch Java Servlet verwendet (heute als Jakarta Servlets bekannt). Natürlich gibt es noch viele weitere Web-APIs dazu.

**Middleware Applications**

Java eignet sich gut für die Vermittlung zwischen den unterschiedlichen Komponenten einer modular verteilten Softwarelösung (Server <-> Clients). So kann mithilfe von Java eine Übersetzungsebene geschaffen werden, die neutral zwischen den Anwendungen vermittelt und die ursprüngliche Komplexität somit verborgen bleiben kann; quasi eine Austausch-API. Man sagt sonst auch „Plumbing“ (also Klempnern) dazu, wobei die Middleware somit „Pipes“ (also Rohre) zur Verfügung stellt, quasi ein Rohrpostsystem.  
Ein Beispiel ist die Apache Tomcat Servlet Engine als Open-Source-Anwendungsserver.

**Server-side Applications**

Datenbanken oder Datenanalyse-Tools, sowie Web- und Fileserver (HTTP/HTTPS + FTP)

**Frameworks und Bibliotheken:**

Java bietet eine Vielzahl quelloffener Frameworks (Rahmen/Gerüste) und vorgefertigter Bibliotheken, die den Einstieg in die Entwicklung vereinfachen.   
<https://struts.apache.org/> - für Webanwendungen (Präsentation und Steuerschicht)

<https://spring.io/> - für Webentwicklung mit Entwicklung mit Java/Java EE (Business)

<https://hibernate.org/> für ORM (objektrationale Abbildung (Objektdaten in Datenbanken speichern und diese später wieder abrufen können). Hibernation meint hier Persistenz.

**Mobile Applications**

Früher wurden mit der Java ME (Micro Edition) Plattform auch Anwendungen für Handys programmiert, bevor Android und iOS sich etabliert hatten. Diese ist nun aber veraltet. Trotzdem wird immer noch Java für die Entwicklung von Android Applikationen genutzt.

# Java verstehen und die Dinge, die es ausmachen

Einstieg über Deshmukh (Seite 8-12 (+32?)) – Grundlagenbegriffe nochmal erklärt:

**Java Development Kit (JDK)**

Enthält alles, was zum Erstellen von Java Programmen gebraucht wird (ohne IDE).

Neben der JRE sind Java Development [Tools](https://docs.oracle.com/javase/8/docs/technotes/tools/) enthalten, sowie der Compiler, Disassembler (Decompiler), Debugger und Analyzer (Java Dependency Analysis Tool - [jdeps](https://g.co/gemini/share/ace473e13374)). Für jedes Zielsystem gibt es eine eigene JDK (**plattformabhängig**).

**Java Runtime Environment (JRE)**

Laufzeitumgebung um die Ausführbarkeit von vorkompilierten Java-Programmen (class- oder jar-files) bereitzustellen. In ihr sind die JVM und die (ebenfalls [vorkompilierten](https://g.co/gemini/share/c06b31f260f9)) Java Libraries enthalten. (Also werden zur Entwicklung nicht nur die Tools der JDK (oben genannt) gebraucht, sondern zum Testen/Ausführen auch die JRE 🡪 Development Tools)

Das Konzept ermöglicht plattformunabhängige Ausführung bereits kompilierter class- oder jar-Files. Die Anpassung, um WORA zu ermöglichen, sind aber systemabhängig.  
Als Teil einer spezifischen JDK ist sie als Konsequenz auch **plattformabhängig**.

Trotzdem muss man damit rechnen, dass bei der Ausführung sehr systemnaher Dinge, wie der Darstellung einer GUI je nach OS/Plattform Unterschiede bestehen werden.

**Java Virtual Machine (JVM)**

Bestandteil der JRE und in seinem allgemeinen Konzept plattformunabhängig (WORA), aber in seiner technischen Umsetzung bzw. konkreten Darreichung hingegen **plattformabhängig** (ausführbare Programmdatei der JVM). Eine virtuelle Maschine im Allgemeinen und somit auch die JVM stellt eine Schnittstelle zwischen dem Hostsystem (Physischer Rechner mit nativem Betriebssystem) und dem Zielsystem (hier Java) dar.

Unterschied zu einer VM als Emulator eines PC/anderes OS/einer Konsole

*Während eine VM wie VirtualBox oder VMware ein komplettes Betriebssystem emuliert, einschließlich seiner Hardware, emuliert die JVM nur die Ausführungsumgebung für Java-Programme. Sie simuliert nicht die gesamte Hardware eines Computers, sondern bietet lediglich eine abstrakte Maschine für die Ausführung von Java-Bytecode.* ([KI-Zitat](https://g.co/gemini/share/2e8144a0ff93))

Die Hardwareabstraktion innerhalb der JVM bietet eine für Java optimierte Umgebung.

Als Kern der Ausführung von Java, ist sie theoretisch unabhängig und kann sogar Java Bytecode ausführen, der aus anderen Quellsprachen gebildet wurde. In der Praxis jedoch ist die JVM ohne JRE (in der sie steckt) bzw. *ohne Java API praktisch nutzlos* (indir. [KI-Zitat](https://g.co/gemini/share/c335358b16de)).

Es soll wohl aber LowEnd-Geräte auf denen die JVM tatsächlich alleinstehend läuft.

**Java Bytecode**

Der Bytecode ist quasi die Maschinensprache der JVM und leicht zu dekompilieren.

Er ist die Zwischenübersetzung, zwischen der Abstraktionsebene des Java-Quellcodes und der Maschinensprache des Systems, welches die JVM hostet und wird interpretiert.

**Java Class Library**

Die auch als Java API bekannte Standard(-klassen-)Bibliothek von Java bietet elementare Funktionsbausteine an, welche dem Entwickler ermöglichen durch Wiederverwendung dieser bereits geschriebenen Klassen kurzen Code zu schreiben, um sich auf das eigentlich zu lösende Problem zu konzentrieren.

**Java Plattform**

Java ME (Micro Edition) - veraltet, oben bereits erwähnt einst für Handys und Kleingeräte

Java SE (Standard-Edition) – von uns/mir verwendet, quasi die Casual-Version. In dieser

enthalten sind die: JVM, Standardbibliotheken (🡪 JRE) und einige Entwicklertools.

Java EE (Enterprise Edition) – jetzt Jakarta EE genannt, weil nun von dieser Organisation verwaltet. Hat zusätzlich zur SE eine Enterprise-Klassenbibliothek und Server JVM.

**Java versions**

//JFC-JA (8 Certification (1Z0-811)) = **J**ava 8 **F**oundations **C**ertified **J**unior **A**ssociate

Java **Version 1** wurde **1996** herausgebracht. (Ich bin nur ein Jahr älter :O)

Die **Version 8**, um die es in der Prüfung geht, wurde **2014** veröffentlicht.

Die aktuelle Version (23.0.1), (Stand 10.01.2025), wurde am 15.10.2024 veröffentlicht.

Im Mai 2025 soll die JDK 24 erscheinen.

Diese Version wird als monumental angesehen und teils noch von einigen Firmen genutzt, weil es dort viele neue Features und bahnbrechende Änderungen gegeben hat.

*Java 8 brachte mit* ***Lambda-Ausdrücken, der Stream API, Default-Methoden in Interfaces, einer neuen Date- und Zeit-API sowie Methodenreferenzen*** *eine funktionale Programmierung nach Java und machte die Sprache damit deutlich ausdrucksstärker und flexibler.* ([KI-Zitat](https://g.co/gemini/share/00bf0c4daf96))

**Java Community Process**

Um zu verhindern, dass jede Firma eigene Java-Variationen entwickelt und diese dann alle zu weit voneinander abweichen, haben Firmen und Entwickler die Organisation Java Community Process ins Leben gerufen. Diese bestimmt Standards, welche alle einhalten müssen, die etwas Fundamentales zu Java beitragen wollen. (Frameworks, APIs, usw.)