

# Java – Erster Überblick

Stephan Karrer

## Designziele von Java, Teil 1

- **Einfach und vertraut:**  
Orientiert an C/C++, aber drastisch gesäubert! Konzepte aus Smalltalk, Ada, ...
- **Objektorientiert:**  
Kapselung, Geheimnisprinzip, Vererbung, Polymorphismus
- **Basis für verteilte Client-Server-Systeme:**  
Bereitstellung umfangreicher Klassen-Bibliotheken und Mechanismen für die Erstellung verteilter Anwendungen
- **Robust:**  
Strenge Typprüfung zur Übersetzungszeit durch den Java-Compiler, keine Zeigerarithmetik, automatische Speicherbereinigung (Garbage Collection), Laufzeitprüfung (Ausnahmebehandlung)
- **Sicher:** Zugriff auf Ressourcen außerhalb der Laufzeitumgebung ist steuerbar

## Designziele von Java, Teil 2

### ■ **Architekturneutral und portabel:**

- verteilt wird Byte-Code, der an der Client-Maschine interpretiert wird (architekturneutral)
- Ein integer in Java ist auf jedem System eine 32-Bit-Zahl (in C nicht ...)
- Unicode für die Codierung von Zeichen, ...

### ■ **Leistungsfähig:**

- Interpreter verlässt sich auf gewisse Prüfungen und wird damit schneller
- Automatische Speicherbereinigung läuft als Hintergrund-Thread mit niedriger Priorität: Speicher ist vorhanden, wenn man ihn braucht
- Schnittstelle zu Maschinencode vorhanden (in C)

### ■ **Multi-Threading-fähig:**

Java unterstützt Threads auf Sprachebene (Thread-Klasse), im Laufzeitsystem (Bausteine zur Synchronisation) und in den Bibliotheken (Thread-sichere Routinen).

## Java Technologie: Spezifikation

- seit 1998 werden die Spezifikationen im sogenannten Java Community Process (JCP) entwickelt
- Es werden 3 verschiedene Plattformen (Frameworks mit bestimmten Umfang) definiert:
  - **JSE** (Java Platform Standard Edition)
  - **JEE** (Java Platform Enterprise Edition)
  - **JME** (Java Platform Micro Edition)
- Oracle (früher Sun) liefert Umsetzungen der Spezifikation als sog. Java Development Kits (JDKs) bzw. Java Runtime Environments (JREs) nur für wenige Plattformen:
  - Hauseigene Systeme: Oracle DBMS, Solaris
  - Linux
  - WinXXX
  - MacOS
- Alle anderen Hersteller implementieren die Spezifikation selbst in ihre Produkte
- Oracle hat seit 2018 an den Lizenzbedingungen für ihre Umsetzung geschraubt
  - Open JDK als Alternative

## Java Standardisierung: JCP, JEP, □JESP



- Die einzelnen Spezifikationen in Form von Java Specification Requests (JSRs) werden durch den Community Process definiert.
- Hier haben sowohl kommerzielle, nichtkommerzielle Organisationen und auch einzelne Personen Zugang.
- Daneben existiert unter dem Open JDK ein Prozess bzgl. des JDK in Form JDK Enhancement -Proposals (JEPs)
- Zukünftig wird die JEE-Version in Form von Jakarta EE durch den Jakarta EE Specification Process (JESP) definiert.

## Entwicklung der SE-Variante

|             |                                     |                                 |
|-------------|-------------------------------------|---------------------------------|
| 1991 – 1995 | Entwicklung der ersten Java-Version |                                 |
| 1           | JSE 1.0                             |                                 |
| 1           | JSE 1.1                             |                                 |
| 2           | JSE 1.2                             |                                 |
| 2000        | JSE 1.3                             |                                 |
| 2002        | JSE 1.4                             |                                 |
| 1           | Java 5.0                            |                                 |
| 2006        | Java 6.0                            |                                 |
| 2011        | Java 7                              |                                 |
| 2014        | Java 8                              |                                 |
| 2017 (Sep.) | Java 9                              | ab jetzt halbjährliche Releases |
| ...         |                                     |                                 |
| 2018 (Sep.) | Java 11 LTS                         |                                 |
| ...         |                                     |                                 |
| 2021 (Sep.) | Java 17 LTS                         |                                 |

## Stand der Dinge

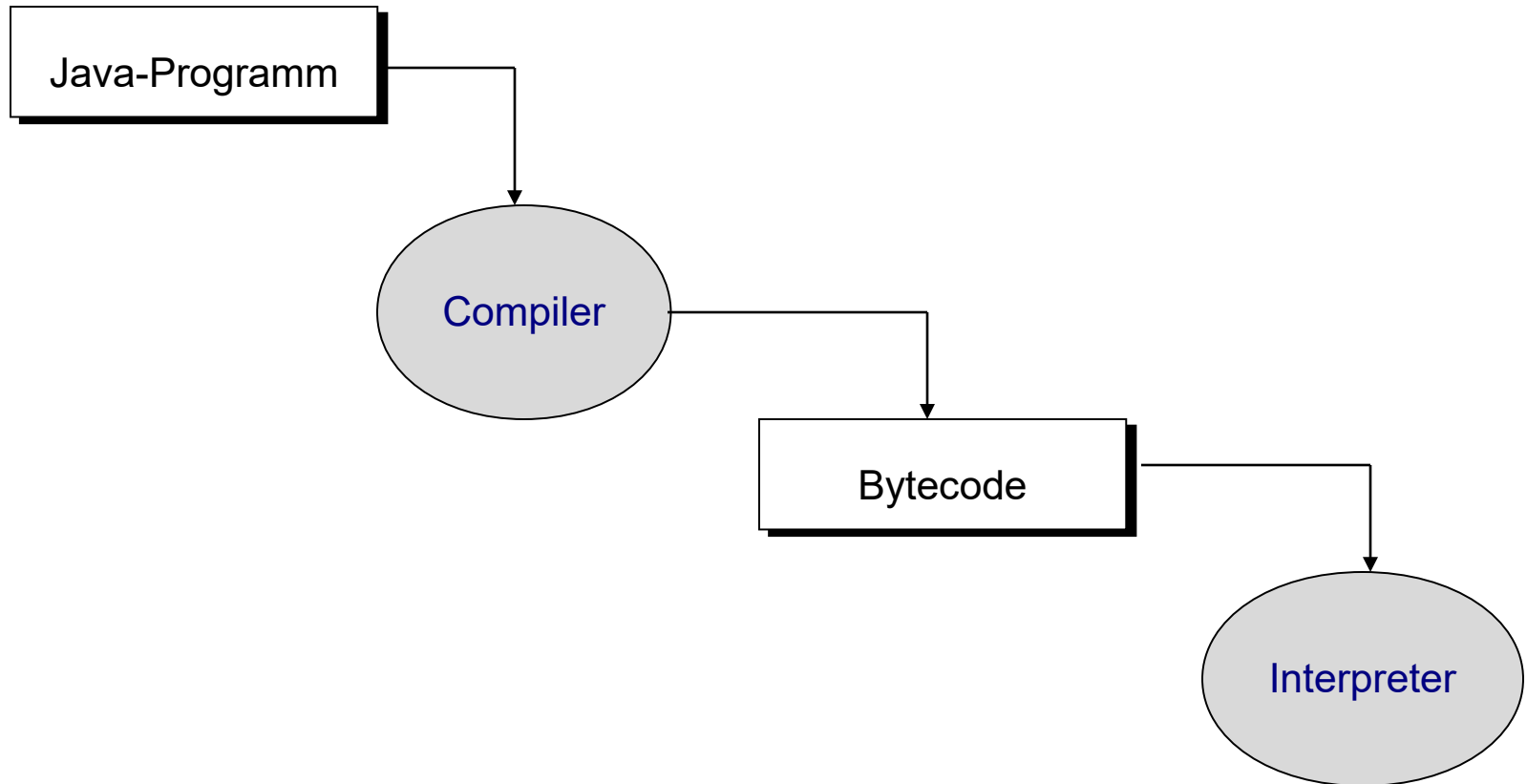
**Aktuelle Java-Versionen:** JSE 17 (ab Sep. 2021)  
JSE 11 (ab Sep. 2019)  
JSE 8.x (ab März 2014)

Jakarta EE 8 ( 2019)  
JEE 8.x (ab Sep. 2017)  
JEE 7.x (ab Mai 2013)

### **Plattformen:**

- MacOS
- Winxxx
- Linux/UNIX
- ....

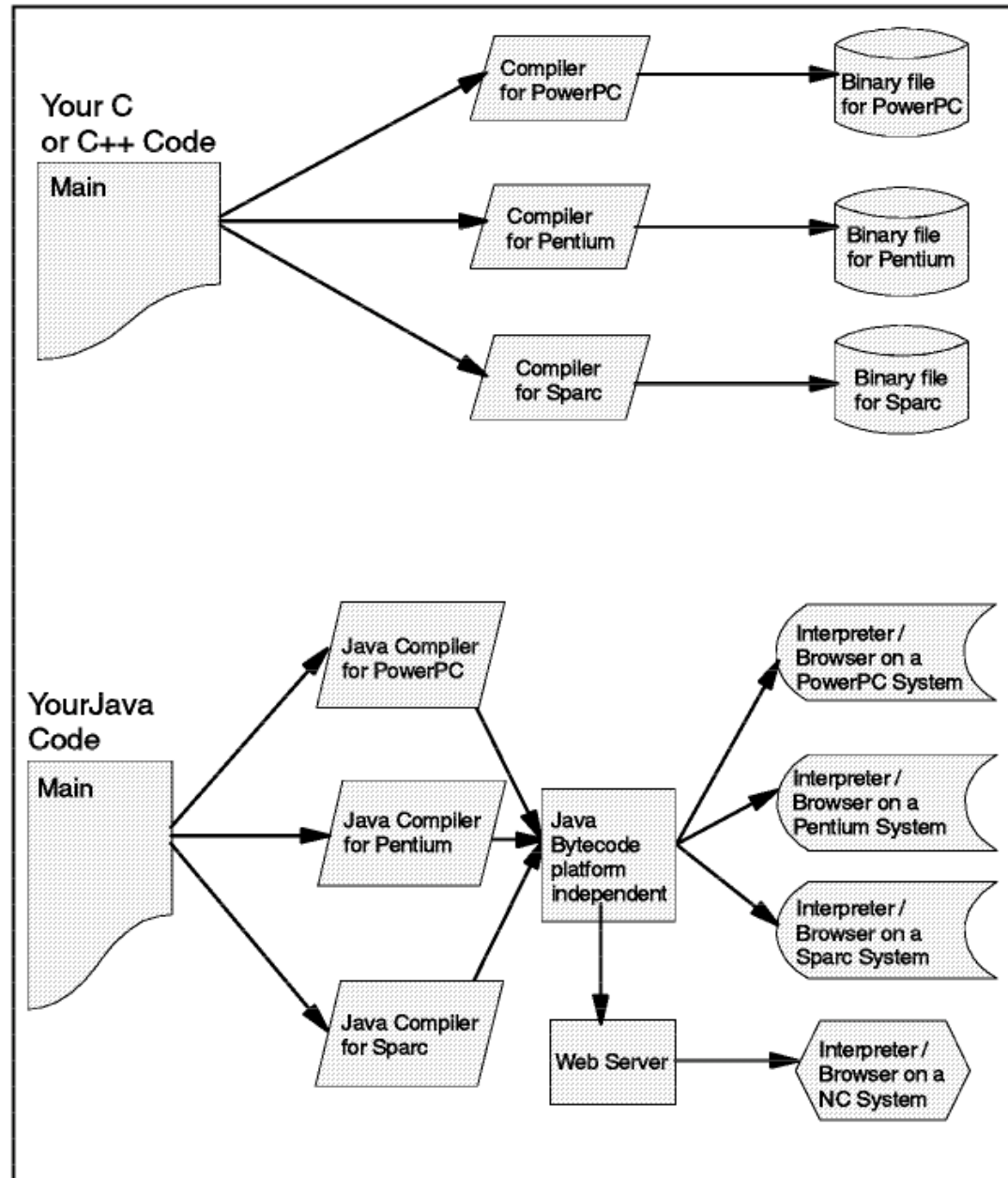
## Grundgedanke bei Java für die Übersetzung und Ausführung





Vergleich:

C- versus  
Java-Compiler



## Java Runtime Environment (JRE)

| Java-Software als Bytecode                               |            |                                   |     |
|--|------------|-----------------------------------|-----|
| Java Laufzeitumgebung<br>(Java Runtime Environment, JRE) |            | Virtuelle Maschine (Interpreter)  |     |
|  |            | Standardisierte Klassenbibliothek |     |
| Windows  | Unix/Linux | Mobiles Gerät                     | ... |

## Dynamisches Binden und Laden

- Statt Linker --> Klassenlader
- Laden ist inkrementell, leichtgewichtig
- Laden ist angepaßt an Netzwerkumgebung
- Laden ist ideal bei häufigen Änderungen
- Java-Compiler löst Referenzen nicht bis zu numerischen Werten (Offsets) auf (Java-Interpreter löst Referenzen einmalig zu Offsets auf beim Einbinden der Klasse)
- Speicheranordnung der Objekte bestimmt der Interpreter (nicht: Compiler). "Einspielen" neuer Klassen

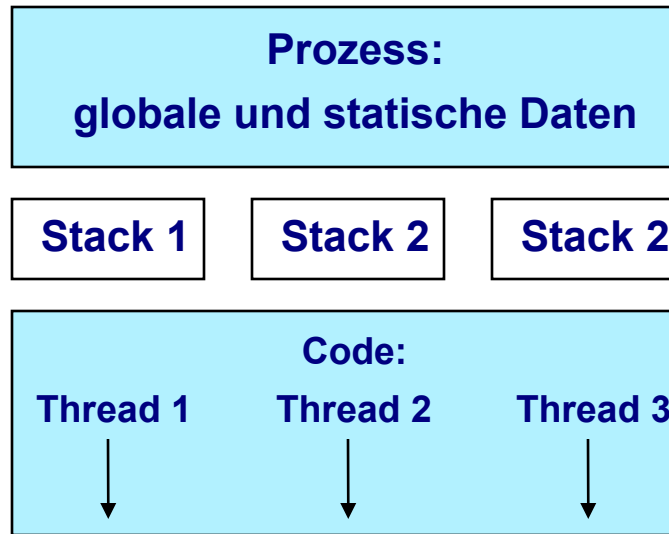
## Informationen zur Laufzeit

- Jedes Objekt enthält Verweis auf ein Class-Objekt
- Zur Laufzeit abfragbar: Klassenname, -typ u.ä.
- Über das Reflection-API können weitere Informationen zu den Klassen und Methoden zur Laufzeit abgefragt werden
  - dies ermöglicht leistungsfähige Debugging-, Monitoring- und Konfigurations-Möglichkeiten
  - kann genutzt werden um dynamisch zur Laufzeit Objekte zu erzeugen und auf Methoden zuzugreifen (die vorab noch nicht bekannt waren)
  - viele Frameworks, insbesondere aus dem JEE-Umfeld, nutzen Reflection

# Speicherverwaltung

- alle Objekte werden dynamisch angelegt (Heap)
- Daten der Basistypen werden ebenfalls dynamisch angelegt
- keine explizite Speicher-Allokierung (wie malloc, calloc o.ä. in C/C++) sondern Objekt-Instantiierung mit new
- Automatische Speicherbereinigung (Garbage Collection)
  - "Wenn keiner mehr darauf verweist: vernichten"
  - "Wenn eine Verweisvariable ihren Gültigkeitsbereich verläßt: Verweis löschen"
- Garbage Collector
  - Thread niedriger Priorität

## Thread-Konzept



### Vorteile:

- effizienter Kontextwechsel
- gut für Serverprozesse
- gut abbildbar auf symmetrische Multiprozessor-Architektur
- effiziente Interthread-Kommunikation

### Nachteile:

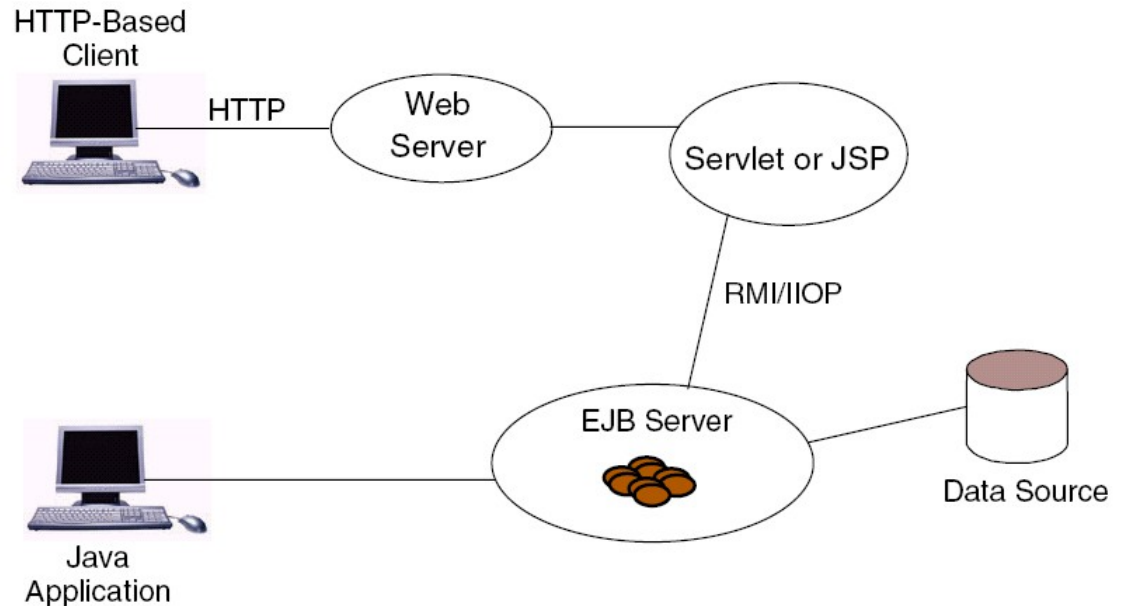
- weniger robust als Multiprozessansatz
- Synchronisation ist eigenes Thema
- Abbildung von "User Level Threads" auf "Kernel Threads" kann auch suboptimal sein
- löst auch nicht die Frage "Was ist parallelisierbar?"

# Thread-Konzept

- Multithreading in die Sprache eingebaut
- Muß nicht notwendigerweise durch das Betriebssystem unterstützt werden, wäre aber günstig
- Zur Verfügung stehen:
  - Klassen Thread, ThreadGroup
  - Operationen start(), interrupt(), join(), yield(), setPriority(), ...
  - Schlüsselwort synchronized
  - Methoden notify() und wait()
- Programmierung vergleichsweise einfach

## Java auf der Serverseite

- Servlets:  
Serverseitiges Äquivalent  
zu Applets
- Java Server Pages:  
Eigene Skriptsprache zur  
Ergänzung
- Enterprise Java Beans:  
Applikationslogik-  
Bausteine die innerhalb  
eines J2EE-Containers  
laufen



- Wichtige Server-Produkte:
  - Apache Web-Server
  - Apache Tomcat: Servlet-Engine
  - RedHat JBoss, Oracle Weblogic, IBM Websphere: Komplette J2EE-Applikationsserver



## Wichtige Entwicklungsumgebungen

- Eclipse (vormals IBM VisualAge, jetzt Open Source)
- NetBeans (vormals Sun/Oracle, jetzt Apache Open Source)
- IntelliJ (IDEA)
- JDeveloper (Oracle)
- Xcode (Apple)
- ...