Überblick

- ❏ Eigenschaften von Java
- ❏ Anfänge des Java-Projekts
- ❏ Design-Ziele
- ❏ Java-Architektur
  - Sprache
  - Java Compiler
  - Java Plattform
- ❏ Dynamisches Laden und Binden
- ❏ Sicherheitsaspekte
- ❏ Unterstützung von Nebenläufigkeit
- ❏ Java im verteilten System

Eigenschaften von Java

Aus dem Hochglanzprospekt …

- **Objektorientiert** & sehr **einfach**
- Ermöglicht **schnellere Softwareentwicklungszyklen**, da **interpretiert**
- Portabel
- Robust
- **Hohe Performance** für graphische Anwendungen
- Anwendungen an Umgebungsänderungen **anpaßbar**, durch dynamisch nachladbare Module
- Anwendungen sind **sicher**, Laufzeitumgebung schützt gegen Viren und andere unbefugte Zugriffe

Anfänge des Java-Projekts

❏ Forschungsprojekt:
Software-Konzepte für *Network Devices* und *Embedded Systems*

❏ Ziel: Ausführungsumgebung mit folgenden Eigenschaften
- klein, zuverlässig, portabel, verteilt, echtzeit-fähig

❏ Ursprüngliche Programmiersprache: C++

❏ Probleme mit C++
- vor allem Sicherheitsprobleme (Zeiger!)

➥ Entwicklung einer völlig neuen Sprache mit Einflüssen aus:
- Eiffel
- SmallTalk
- Objective C
- Cedar/Mesa

Design-Ziele

❑ Traditioneller Softwareaufbau und -vertrieb wird immer problematischer

- unterschiedliche Betriebssysteme (Unix, Windows, MacOS, …)
  → unterschiedlicher Aufbau von Softwarepaketen

- unterschiedliche Hardwarearchitekturen
  → unterschiedliche Binärdateien

❑ Java: Sprache und Plattform für:

Sichere, **hochperformante**, **sehr robuste**
Anwendungen für
verschiedene Plattformen in
heterogenen, **verteilten Netzen**

❑ Java: einfach, objektorientiert, leicht erlernbar

Java-Architektur

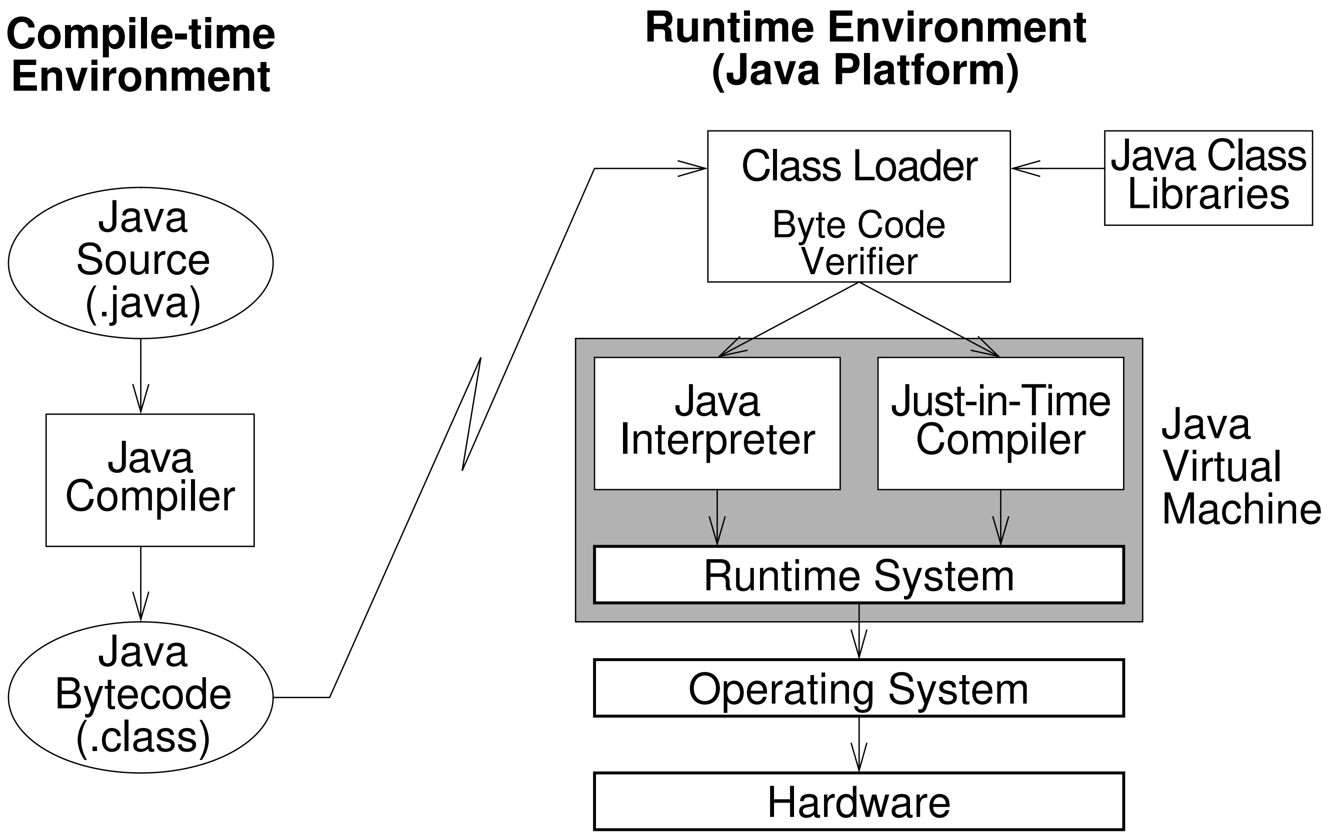
- ❏ Sprache
- ❏ Java Compiler
- ❏ Java Plattform
  - Klassenbibliotheken
  - Virtuelle Maschine
  - Basissystem

Java — Sprache

angelehnt an C++, aber deutliche Unterschiede

- keine Zeiger
- echte Felder mit Feldgrenzenüberprüfung, echte String-Objekte
- kein Preprozessor
- keine Funktionen, Strukturen und Unions
- Speicherfreigabe durch Garbage Collection
- integriertes Threadkonzept
- keine Mehrfachvererbung von Klassen
- Konzept für Schnittstellendefinitionen (Typen), unabhängig von Klassen

Java Environment



Java Compiler

- Java-Quellcode → Java-Bytecode
  - Bytecode-Generierung
  - Erzeugung von symbolischen Namen für Methoden, Klassen, Instanzvariablen, Typinformation, …
    - ➥ Unterstützung von dynamischer Bindung auf der Basis von Namen sowie Sicherheitsüberprüfungen zur Laufzeit

- Prüfung auf Syntaxfehler

- Strenge, statische Typprüfung

- wenn notwendig, automatische Rekompilierung importierter Klassen (.java-Datei neuer als .class-Datei, wie bei Make)

Java Bytecode

- Java Quelle

```
class C {
  public void m1(int a){}
  public static void main(String args[]) {
    C o = new C();
    o.m1(3);
    o.m1(12);
  }
}
```

- Byte-Code

```
Method void main(java.lang.String [])
   0 new #2 <Class C>
   3 dup
   4 invokespecial #4 <Method C.<init>()V>
   7 astore_1
```

C o = new C();

```
   8 aload_1
   9 iconst_3
  10 invokevirtual #5 <Method C.m1(I)V>
```

o.m1(3);

```
  13 aload_1
  14 bipush 12
  16 invokevirtual #5 <Method C.m1(I)V>
```

o.m1(12);

```
  19 return
```

Klassenbibliotheken = Java Packages

bilden Grundlage für verschiedene APIs
➥ **Application Programming Interfaces**

- Packages dienen gleichzeitig der Strukturierung des Namensraums für Klassen

❑ Java Core API

- Minimale Plattform, die in jeder Java-Umgebung verfügbar sein muß (ausgenommen Personal Java und Embedded Java Plattformen)

❑ Java Standard Extension API

- Zusätzliche Schnittstellen für bestimmte Anwendungsbereiche, für die von JavaSoft ein API-Standard veröffentlicht wurde

❑ PersonalJava API

- für netzwerkfähige Anwendungen auf kleinen Geräten (hand-held computers, smart phones, set-top boxes, game consoles, …)

❑ EmbeddedJava API

- für Massenprodukte im Embedded-Systems-Markt: Mobiltelephone, Prozeßkontrolle, Netzwerk-Router, … — geringe Speicher-Anforderung!

Java Core API

- ❏ Minimale Plattform, die in jeder Java-Umgebung verfügbar sein muß
  – Network Computer
  – PC
  – Workstations

- ❏ wesentliche Java packages:

• java.lang	Sprachunterstützung (threads, integer, ...)
• java.lang.reflect	Zugriff auf Klassen- bzw. Objektstruktur
• java.io	E/A-Funktionen (Files, Streams, ...)
• java.net	Netzwerk (Sockets, URLs, ...)
• java.rmi	Remote Method Invocation
• java.awt	Windows
• java.applet	Unterstützung für Applets
• java.beans	Unterstützung für Komponentenarchitektur
• java.security	Sicherheitsmechanismen (z.B. ACLs)
• java.sql	Datenbank-Schnittstellen (JDBC)

Java Standard Extension API

- ❏ Anwendungsschnittstellen für spezielle Anwendungsbereiche
    - definierte Schnittstellen für bestimmte Mechanismen
    - unterschiedliche Implementationen in Abhängigkeit von der jeweiligen Plattform
    - offen für Erweiterungen
    - Beispiele:
        - Java Server
        - Java Commerce
        - Java Management
        - Java Media
            - Media Framework
            - Collaboration
            - Telephony
            - Speech
            - Animation
            - 3D

Java Virtual Machine

- ❏ Schlüssel für Architekturunbhängigkeit
- ❏ derzeit in ANSI-C implementiert

Interpreter

- ❏ Bytecode interpretieren
  - Ausführung von Maschineninstruktionen
  - Ausführung von komplexen Funktionen des Laufzeitsystems

- ❏ Sicherheitsüberprüfungen
  - Typkompatibilität
  - Einhaltung von Feldgrenzen
  - Einhaltung von Stackgrenzen
  - Null-Referenzen

Java Virtual Machine (2)

Laufzeitsystem

❑ Abbildung von Bytecodes auf komplexere Funktionen

- Speicher anfordern
- Klassen laden
- Objekte instantiieren
- Methoden ausführen

❑ Garbage Collection

❑ Erweiterung der virtuellen Maschine durch *native Methoden*

- Implementierung in C, C++, Modula, Assembler, etc.
- wird in Form von dynamischen Bibliotheken der Java-Maschine verfügbar gemacht
- Schnittstelle über normale Java-Klassen, *native* Methode ist deklariert, aber nicht implementiert
- Beispiele: Netz- und Dateisystemschnittstelle, Motif-Schnittstelle

Java Virtual Machine (3)

Optimierungen der Java VM

- ❏ Just-in-Time Code Generation (➞ Faktor 10)
- ❏ Predication, Speculation
- ❏ Dynamic inlining & Loop optimizations
- ➥ Ziel: schneller als derzeit bestes C++

Basissysteme für die Java-Plattform

WWW-Browser

- HotJava
- Netscape

Java Interpreter als Programm auf einem herkömmlichen BS

- vergleichbar mit Shell oder Perl
- Interpreter-Start z. T. in das Betriebssystem integriert (exec-Systemaufruf unter Linux)

JavaOS

- Betriebssystem für Java-Anwendungen, in Java geschrieben

Basissysteme für die Java-Plattform (2)

JavaChips

➥ **picoJava** **microJava** **UltraJava**

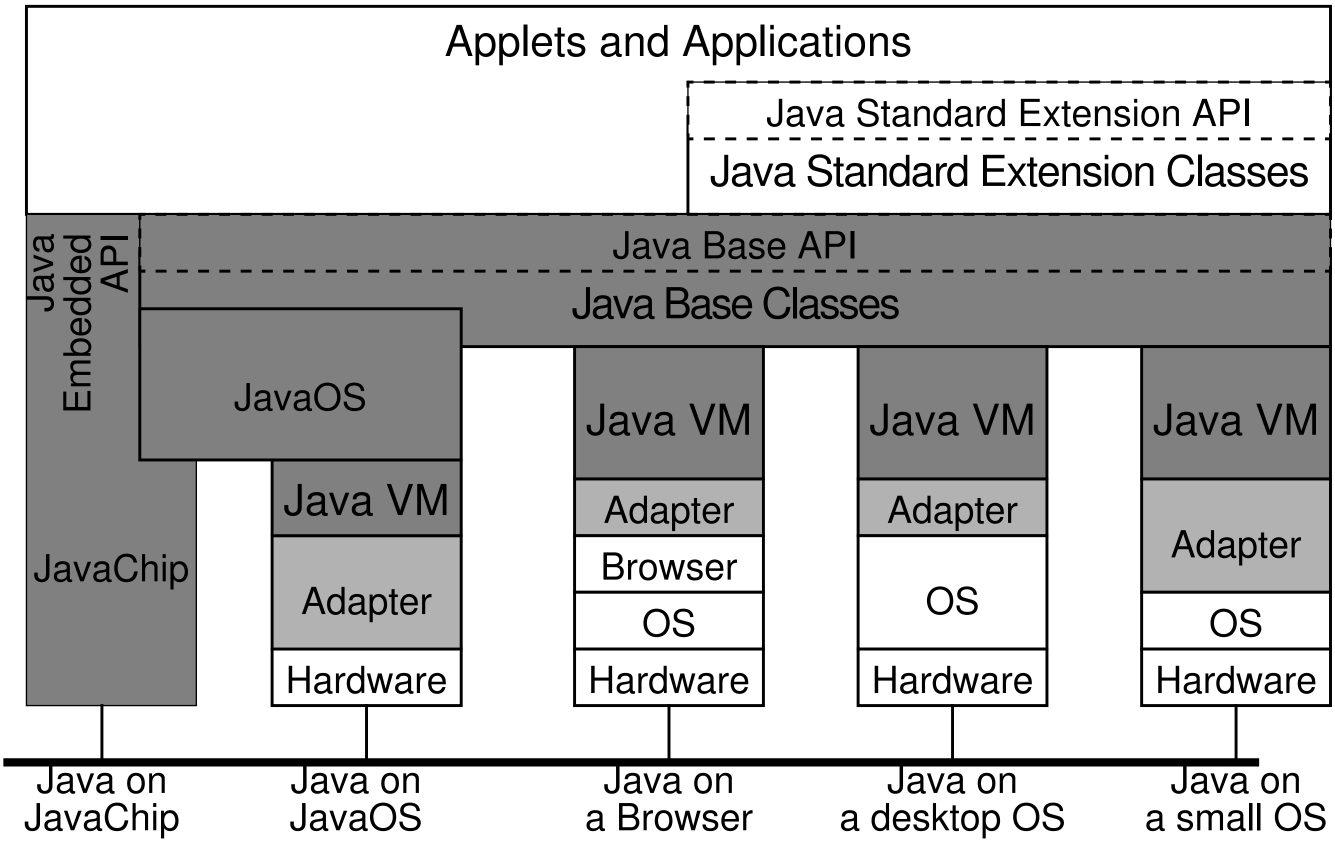
- ❏ für Java-Umgebung optimierte Prozessoren

- ❏ Java VM und Java Embedded API in Hardware implementiert

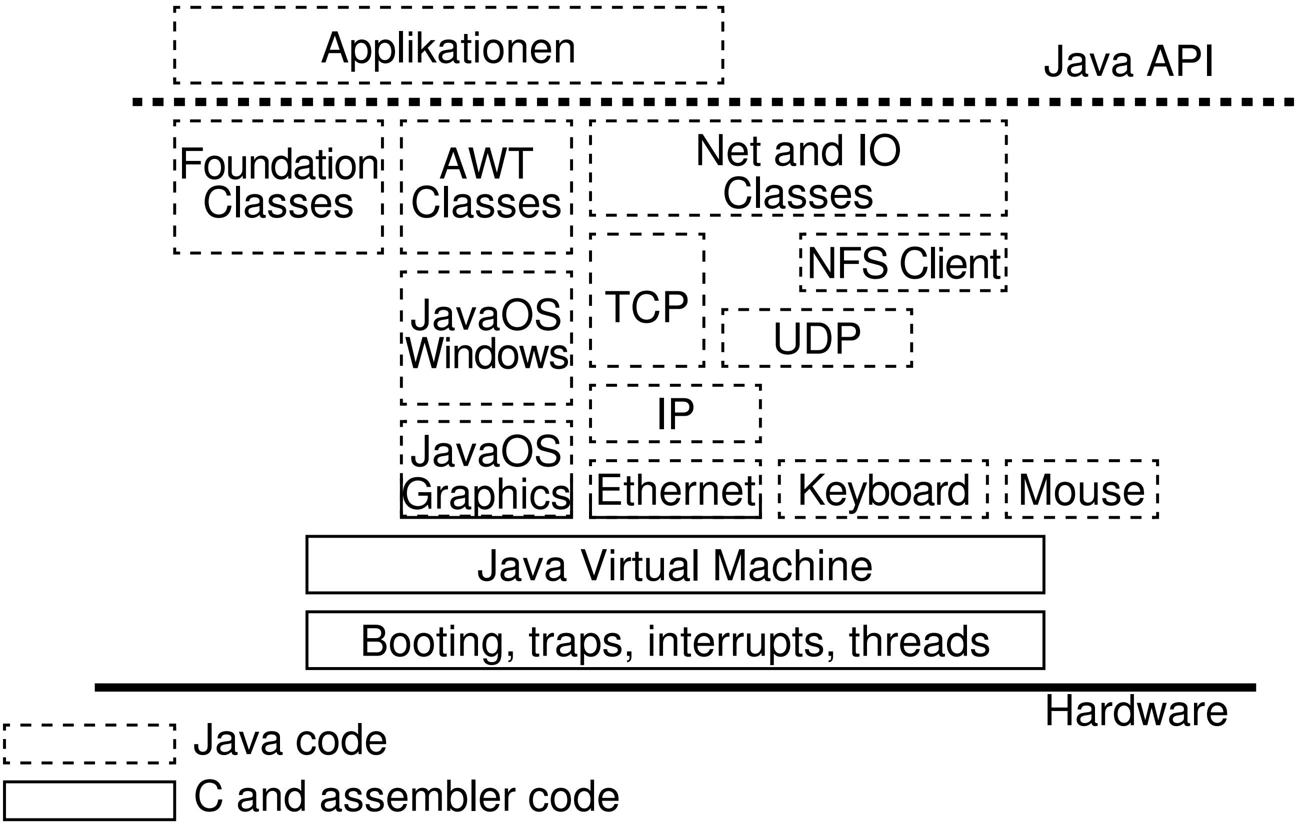
- ❏ JavaOS im RAM zur Erweiterung der Funktionalität

- ❏ Einsatzgebiete
  - picoJava (< 25$): Controller im Embedded-Systems-Markt
  - microJava (25 - 100 $) Netzwerkorientierte Komponenten, Steuerungen, Spielsysteme, einfache Terminals
  - UltraJava 64bit Prozessor für komplexe Graphikanwendungen, Multimedia-Bereich

Java Plattform — Überblick



JavaOS Architektur



Dynamisches Laden und Binden

❏ keine Binde-Phase

- Klassen werden bei Bedarf dynamisch durch den *Class Loader* zu einer Anwendung geladen
- anwendungsspezifische *Class Loader* möglich
- Klassen können über Netzwerk oder aus dem lokalen Dateisystem geladen werden

❏ *Fragile Superclass Problem*

- statisch bindende Umgebungen (wie C++) haben Probleme, wenn einzelne Klassen geändert werden (z. B. neue Methoden)

  ➥ Anwendung muß komplett neu übersetzt werden, weil Speicher-Layout und Methodentabellen nicht mehr stimmen!

- der Java-Compiler erzeugt nur symbolische Referenzen, Speicher-Layout und Referenzen-Auswertung erfolgen erst zur Laufzeit

  ➥ *Fragile Superclass Problem* wird vermieden (wichtig für Änderungen in Bibliotheken!)

Sicherheitsaspekte

❏ Speicherverwaltung

- Speicherverwaltung erfolgt nicht durch das Programm, sondern durch den Interpreter zur Laufzeit (Speicherfreigabe durch Garbage Collection)
- keine Zeiger → kein direkter Zugriff auf Speicherbereiche möglich
→ keine ungültigen Zeiger
- Feldgrenzen werden bei Zugriffen überprüft
- Referenzen werden bis zur Laufzeit als symbolische Verweise verwaltet und können zur Laufzeit nicht manipuliert werden

❏ Bytecode Verifier

- Überprüfung des Bytecodes nach dem Laden
- verifizieren, daß der Bytecode den Richtlinien des Java-Compilers entsprechend aufgebaut ist
  - statische Typprüfung
  - korrekter Zugriff auf Objektinhalte (private, public, protected)
  - operand stack overflows & underflows

Sicherheitsaspekte (2)

❏ Laufzeitprüfungen

❏ Sicherheitsmechanismen für Einsatz von Java im Internet

- Unterschied zwischen Applet und Applikationen
  - Applikation: wird normalerweise aus lokalem Dateisystem geladen und von einem Java-Interpreter ausgeführt (kann aber auch über Netz kommen - gefährlich!)
  - Applet: wird über WWW geladen und in einem Browser ausgeführt
- Applikationen haben gleiche Freiheit wie “normale” Anwendungen (C++)
  - aber Gefahr, wenn Klassen über Internet geladen werden!
- Rechte von Applets hängen von Browser-Konfiguration ab:
  - keine Netzzugriffe, keine Dateisystemzugriffe
  - eingeschränkte Netzrechte: Verbindungen nur zu dem Rechner, von dem das Applet geladen wurde, keine Dateizugriffe (Netscape)
  - beliebige Zugriffe

Unterstützung von Nebenläufigkeit

- ❏ Thread-Konzept und Unterstützung zur Steuerung von Koordinierungsmechanismen sind in Java integriert

  ➥ nicht orthogonale, nicht uniforme Sprache bzgl. Nebenläufigkeit

- ❏ Erzeugung von Threads über Thread-Klassen

- ❏ Definition der auszuführenden Methode über spezielle Klassen, die ein Interface **`Runnable`** implementieren (Methode **`run`**)

  - Thread erzeugen = Thread-Objekt + “run-Methoden”-Objekt instantiieren
  - Thread starten = Methode **`start`** an Thread-Objekt aufrufen,
“run-Methoden”-Objekt übergeben
→ Methode **`run`** wird nebenläufig ausgeführt

- ❏ Koordinierungsmechanismen

  - Monitore: exclusive Ausführung von Methoden eines Objekts
  - Conditions: gezieltes Freigeben des Monitors und Warten auf ein Ereignis

Java im verteilten System

- ❏ Java RMI (*Remote Method Invocation*) zur Unterstützung von Interaktionen mit Objekten in anderen virtuellen Java Maschinen (lokal oder über Netz!)

- ❏ RMI auf Java beschränkt, daneben aber auch CORBA-konforme Anbindungen (im Rahmen von Sun NEO oder ORBIX)

- ❏ Objekte müssen ein Interface **`Remote`** anbieten, um aus anderen Java Maschinen ansprechbar zu sein
  - Klassen sollten von der Klasse **`RemoteServer`** erben, um die Mechanismen für Remote-Zugriffe zu erhalten

- ❏ Remote-Zugriffe werden über Stub-Objekte vermittelt
  - Dynamic Stub Loading!

- ❏ Problem bei Parameterübergabe:
  non-remote objects werden *by copy* statt *by reference* übergeben

- ❏ verteilte Garbage Collection