Wo sind wir?

- **Java-Umgebung**
- Lexikale Konventionen
- Datentypen
- Kontrollstrukturen
- Ausdrücke
- Klassen, Pakete, Schnittstellen
- JVM
- Exceptions
- Java Klassenbibliotheken
- Ein-/Ausgabe
- Collections
- Threads
- Applets, Sicherheit
- Grafik
- Beans
- Integrierte Entwicklungsumgebungen

Java

- Entstanden aus einem Projekt (Green) bei SUN für Consumer Electronics, Set-Top-Boxen usw. in den frühen 1990er Jahren, für das die Programmiersprache Oak entwickelt wurde, aus der wiederum Java entwickelt wurde
- 1995 erster Sprachstandard (von SUN)
- Entwicklung von Java wird nach wie vor von SUN dominiert
- Rasante Entwicklung zu Beginn eng gekoppelt mit dem Einsatz in Browsern (Netscape)
- Applets und Anwendungen
- Freies Java Development Kit (JDK) u.a. von SUN
- Java hat nichts aber auch auch gar nichts zu tun mit JavaScript

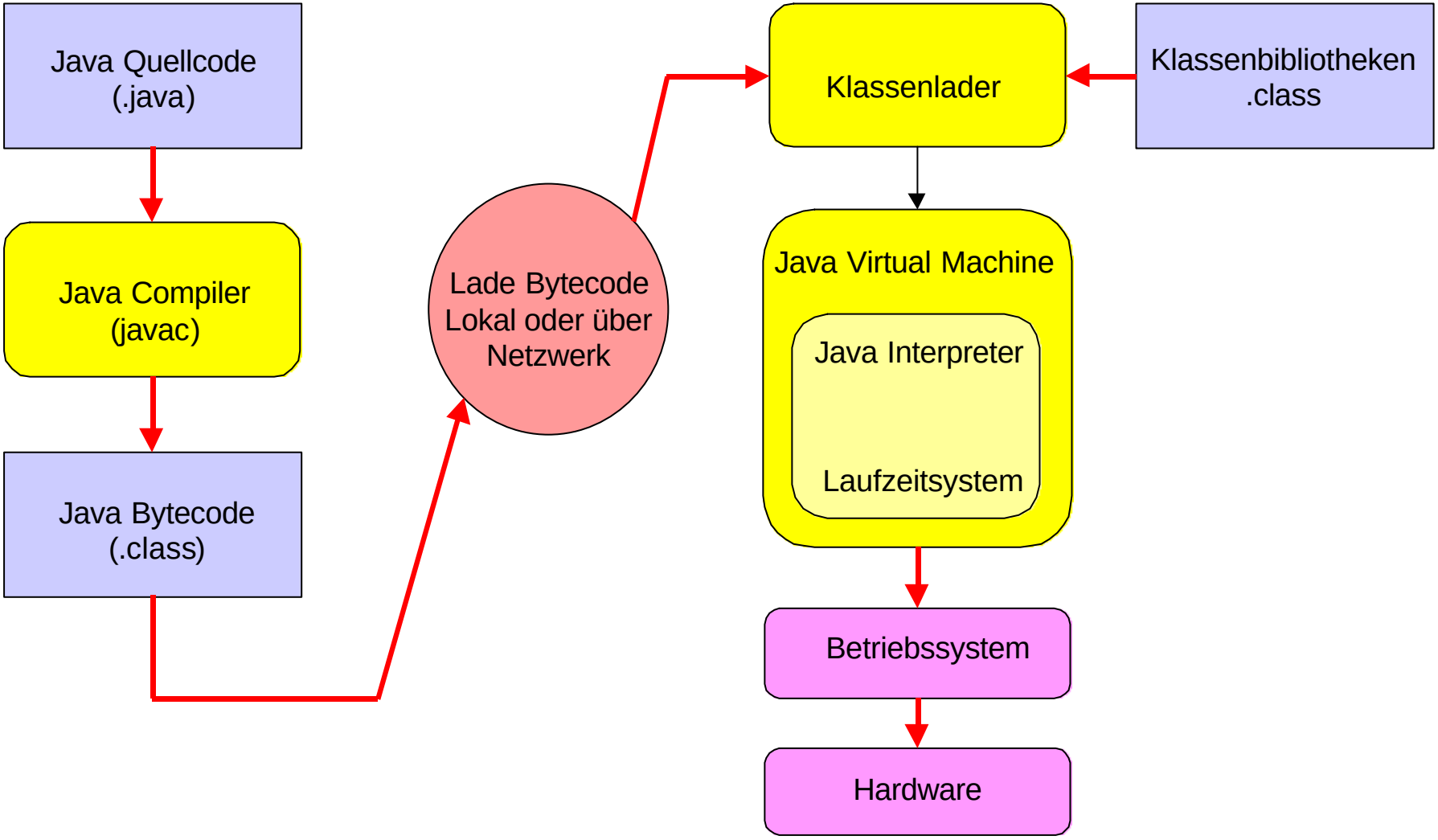
Eigenschaften von Java

- Vorteile gegenüber anderen Sprachen:
  - Saubere Sprachdefinition
  - Feste Längen für alle Datentypen
  - Exakt definierte Arithmetik
  - Automatisches Speichermanagement
  - Umfangreiche Software-Bibliothek
  - Plattformunabhängig durch JVM (Java Virtual Machine)
  - Sicherheitskonzept (Bytecode-Verifier, Signed Applets, feingranulare Zugriffsrechte)
  - Verteilte Anwendungen
  - Integrierte Threads
- Nachteile:
  - Oft langsamer in der Ausführung (Erklärungen später)
  - Inkompatible Versionen des JDK im Umlauf

Java Umgebung

Entwicklungsumgebung

Laufzeitumgebung



JDK 1.3

- JDK: Java Development Kit von SUN
- Java 2 Standard Edition (J2SE)
- Offizieller Name: SDK 2, Standard Edition, Version 1.3
  (SDK steht für Software Development Kit)
- Frei verfügbare Entwicklungs- und Laufzeitumgebung für Windows, Solaris, Linux (u.a.)
- Derzeitige Version: 1.3
- Auf allen Pool-Rechnern installiert
- Siehe WWW-Seite zur Veranstaltung für Download des JDK
- JRE: Java Runtime Environment
  Enthält nur die Java-Komponenten, die man zur Ausführung von Java-Programmen benötigt (z.B. kein Compiler)
- Java Plug-in
  JRE zur Integration in WWW-Browser (Netscape, Microsoft Internet Explorer)
- Java 2 Enterprise Edition (J2EE) (für Server)
- Java 2 Micro Edition (J2ME, z.B. für PDA's)

javac

Java Compiler

Aufruf:

```
javac [options] source-files
```

source-files:

Beliebige Anzahl von Java-Dateien, die die Endung .java haben müssen (Endung mit angeben). Mehrere Dateien werden durch Leerzeichen getrennt.

Wichtigste options:

`-g`	Debugger-Informationen erzeugen
`-O`	Optimierung einschalten
`-verbose`	Detaillierte Informationen, was der Compiler macht
`-deprecation`	Warnung bei Nutzung veralteter API's ausgeben
`-classpath path`	Pfadangabe zu Benutzerklassen
`-d directory`	Directory, wo .class-Dateien abgelegt werden sollen

Beispiele:

```
javac MeinTest.java
javac *.java
javac -O -classpath /home/benutzer/classes MeinTest.java
```

java

Java Laufzeitumgebung (inkl. JVM)

Aufruf:

```
java      [options] class     [args]
java -jar [options] jar-Datei [args]
```

class oder jar-Datei:

Übersetzte Datei, die die Endung .java bzw. .jar haben muss (Endung nicht mit angeben).

Wichtigste options:

`-verbose`	Detaillierte Informationen, was die JVM macht
`-classpath path`	Pfadangabe zu Benutzerklassen (durch ; getrennte Liste von Verzeichnissen und/oder zip/jar-Dateien)
`-Dname=value`	Setzen von Properties (Umgebungsvariablen)
`-Xms=size`	Setze anfängliche Heap-Größe
`-Xmx=size`	Setze maximale Heap-Größe

Beispiele:

```
java MeinTest
java -classpath /local/classes;./klassen.jar -Xmx=1000000 DeinTest
```

JIT

JIT = Just-In-Time-Compiler

Heutige JVM's erkennen bei der Interpretierung von Java Bytecode Code-Bereiche, die z.B. wiederholt ausgeführt werden (Beispiel: Schleifenrumpf).

In diesem Fall erzeugen Sie Maschinencode des Rechners und ersetzen den Bytecode dieses Codebereiches durch einen Aufruf des Maschinencodes, der schneller ausgeführt werden kann.

Beispiel:

```
class MeineKlasse {

    void methode(int i) { ... }

    public static void main(String[] args) {
        for(int i=0; i<100; i=i+1)
            methode(i);
    }
}
```

jdb

Java Debugger

Aufruf:

```
jdb       [options] class      [args]
```

options:

`-launch`	Startet automatisch eine VM
`-sourcepath path`	Suchpfad für Source-Dateien

class:

Übersetzte Datei, die die Endung .java bzw. .jar haben muss (Endung nicht mit angeben).

Beispiele:

```
jdb -launch MeinTest
```

Der Debugger kann nur mit Klassendateien arbeiten, die mit -g übersetzt wurden.

Kommandos des Debuggers jdb

Innerhalb der Debugger-Umgebung gibt es eigene Debugger-Kommandos. Die wichtigsten davon sind:

- exit oder quit
  Beenden des Debuggers
- load Klassenname
  Lädt angegebene Klasse in Debugger
- classes
  Listet alle geladenen Klassen auf
- list [Zeilennummer]
  list Methodenname
  Auflisten der Source-Datei
- print Bezeichner
  Gibt Wert (Inhalt) des Bezeichners aus
- where
  Aufrufhierarchie
- stop at class:line
  stop in class.method
  Breakpoint setzen an angegebener Stelle
- clear
  Löschen von Breakpoints
- cont
  Fortsetzen der Ausführung an Breakpoint
- catch [Exception-Klasse]
  ignore Exception-Klasse
  Breakpoint für Exception setzen/löschen
- next
  Führe nächsten Schritt aus
- step
  Führe nächsten Schritt aus (in Methoden rein)

Das Kommando help gibt die Liste aller Kommandos aus.

Beispiel eines Aufrufs

```
C:\java>jdb -launch fakultaet
...
main[1] stop in fakultaet.main
Deferring breakpoint fakultaet.main.
It will be set after the class is loaded.
main[1] cont
> Set deferred breakpoint fakultaet.main
Breakpoint hit: thread="main", fakultaet.main(), line=9, bci=0
  9         long n = 20;
main[1] next
main[1] Step completed: thread="main", fakultaet.main(), line=11, bci=4
  11    for(i=0; i<20; i++)
main[1] next
main[1]
Step completed: thread="main", fakultaet.main(), line=12, bci=9
  12        System.out.println("Fakultät von " + i + " ist " + fac(i));

main[1] step
main[1]
Step completed: thread="main", fakultaet.fac(), line=3, bci=0
  3         if(n==0)

main[1] where
  [1] fakultaet.fac (fakultaet.java:3)
  [2] fakultaet.main (fakultaet.java:12)
```

Richtlinien für Java-Dateien

- http://java.sun.com/docs/codeconv/ (siehe WWW-Seite zur Veranstaltung)
- Einheitlicher Aufbau und Aussehen von Java-Dateien
- Pro Zeile nur eine Variable deklarieren

```
int i;
int j;
```

- Pro Zeile nur eine Anweisung
- Alle Variablendeklarationen zu Beginn des zugehörigen Blocks

```
{
    int i;
    i = 1;
    int j = 2;   // falsch!
}
```

- Methoden haben die Form:

```
int methode(int arg) {
    ...
}
```

- 4 Leerzeichen einrücken (Emacs rückt z.B. im Java-Modus jede Zeile automatisch richtig ein durch Drücken von TAB in der Zeile)

Richtlinien für Namen

- Variablen:
  Variablennamen beginnen mit einem Kleinbuchstaben gefolgt von Klein- oder Großbuchstaben. Teilwörter beginnen mit Großbuchstaben.
  Beispiel:

```
int      i;
double   grosseZahl;
```

- Konstanten:
  Namen von Variablen, die als konstant deklariert sind, werden komplett in Großbuchstaben geschrieben mit _ zwischen Teilwörtern.
  Beispiel:

```
static final int BILDSCHIRM_BREITE = 100;
static final float PI = 3.1415;
```

- Methoden:
  Methodennamen sollten Verben sein, die mit einem Kleinbuchstaben beginnen und Großbuchstaben zu Beginn von Teilwörtern haben.
  Beispiel:

```
int startenMotor()
void kassierenGeld(int summe)
```

Richtlinien für Namen

- Klassen

  Klassennamen sollten Hauptwörter in Klein- und Großbuchstaben sein. Der erste Buchstabe von Teilwörtern sollte groß geschrieben sein.

  Beispiel:

```
class Zoo {...}
class LoewenGehege {...}
```

- Paketnamen

  Paketnamen bestehen nur aus Kleinbuchstaben.

  Beispiel:

```
package meininternespaket;
package de.fh-bonn-rhein-sieg.inf.meintollespaket;
```

Kommentare

Es gibt drei Arten von Kommentaren in Java:

- Herkömmlicher Kommentar:

```
/* beliebiger Text,
   der über mehrere Zeilen gehen kann und z.B. * oder * / (nicht zusammen
   geschrieben!) enthalten kann */
```

- Einzeiliger Kommentar:

```
// Beliebiger Text bis zum Ende der Zeile
```

- Dokumentationskommentar

```
/** Dokumentationstext  (mehr dazu gleich) */
```

Kommentare können nicht geschachtelt werden!

Dokumentationskommentare

Dokumentationskommentare sind spezielle Kommentare, aus denen sich mit Hilfe des Werkzeugs javadoc HTML-Dokumentation in standardisierter Form erstellen lässt.

Solche Kommentare können plaziert werden direkt vor:

- Klassen- und Schnittstellendeklarationen
- Methodendeklarationen
- Konstruktordeklarationen
- Datenfelddeklarationen (d.h. Variablendeklarationen in Klassen)

Regeln für Dokumentationskommentare

Regeln:

- Kommentare können über mehrere Zeilen gehen
- Einleitende Sterne (*) werden dabei ignoriert
- Leerzeichen und Tabs vor den Sternen werden ab der zweiten Zeile ebenfalls ignoriert
- HTML-Befehle im Kommentar sind erlaubt (aber besser nicht nutzen)
- Der erste Satz (bis zum ersten Punkt) sollte eine kurze Zusammenfassung enthalten.
- Nach dem ersten Satz folgt der allgemeine Beschreibungsteil.
- Es gibt ausgezeichnete Abschnitte, die mit einer Markierung @Schlüsselwort beginnen.

Beispiel:

```
/**
  * Newton's Verfahren zur Berechnung der Wurzel einer Zahl. Wir verwenden
  * hier ein iteratives Verfahren, um die Wurzel einer Zahl zu bestimmen.
  * @author Rudolf Berrendorf
  */
```

Markierungen

- @author Namenstext
  Kennzeichnet den Autor dieser Software
- @version Versionstext
  Angabe einer Versionsnummer (Release-Nummer)
- @see Klassenname
  @see Klassenname#Member
  Erzeugt einen Hyperlink zur Klassenbeschreibung
- @param Name Beschreibung
  Beschreibung zu einem formalen Parameter einer Methode
- @return Beschreibung
  Beschreibung zum Rückgabewert einer Methode
- @exception voll-qualifizierter-Klassenname Beschreibungstext
  Erzeugt einen Hyperlink zur Exception in voll-qualifizierter-Klassenname
- @since Seit-Text
  Seit wann das Interface unterstützt wird (Seit-Text meist eine Release-Nummer)
- @deprecated Text
  Hinweis, dass diese Klasse/Methode nicht mehr genutzt werden sollte

Beispiel

```
/**
 * Implementierung für das Newton-Verfahren zur Wurzelberechnung.
 * @author Rudolf Berrendorf
 * @version 1.0
 * @see java.lang.Math#sqrt(double)
 */
public class MeineKlasse {

    /**
     * maxiter enthält die maximale Anzahl an Iterationen.
     */
    int maxiter;

    /**
      * Berechnet die Wurzel einer Zahl.
      * @return Wurzelwert der Zahl
      * @param wert Zahl, von der Wurzel berechnet werden soll
      */
    public double wurzel(double wert) {
        // ...
    }
}
```

javadoc

Aus einer Java-Datei mit Dokumentationskommentaren lassen sich mehrere HTML-Dateien erzeugen, die die Dokumentation in lesbarer Form zur Verfügung stellen.

Aufruf:

```
javadoc [options] [packagenames] [sourcefiles] [classnames]
```

Einige Optionen:

`-public`	Nur Sachen zeigen, die public sind
`-protected`	Nur Sachen zeigen, die protected sind (Default)
`-package`	Nur Package-Sachen zeigen
`-private`	Alle Sachen zeigen
`-sourcepath dirlist`	Wo Source-Dateien zu finden sind
`-classpath dirlist`	Wo Klassen zu finden sind
`-d directory`	Wo die Ausgabe gespeichert werden soll