

Objektorientiertes Programmieren mit Java

Eine Vorlesung an der Universität Bayreuth im WS 2001/2002

WWW-Seite der Vorlesung

http://www.uni-bayreuth.de/departments/math/~rbaier/lectures/java_multimedia_ws2001_02/

Nur die mit führendem “*” gekennzeichneten Abschnitte beinhalten Informationen, alle anderen Abschnitte verweisen auf die Vorlesung.

Bei diesen Seiten handelt es sich um eine Ergänzung mit einigen Details, die die ergänzenden Seiten der Vorlesung weiter vertiefen sollen.

Inhaltsverzeichnis

0	* Einführung	3
0.1	* Erste Beispiele	3
0.2	* Literatur	3
0.3	* Was ist Java	3
0.4	* Geschichte	5
0.5	* Ziele	12
0.6	* Java-Applikationen und -Applets	17

0 * Einführung

0.1 * Erste Beispiele

0.2 * Literatur

0.3 * Was ist Java

Java gibt es in verschiedenen Distributionen/Editionen, je nach Aufgabenzweck.

Java™ 2 Platform, Micro Edition (J2ME™) vgl. Homepage von J2ME

Java-Version für Haushaltselektronik und Verbrauchsgegenstände (128 - 512 KB Speicher oder mehr)

besteht aus einer Virtual Machine und einer Menge von APIs Diese bestehen aus Konfigurationen und Profilen. Laut einer Umfrage sollen von Entwicklern für drahtlose Geräte 33 % JME benutzen, 24.5 % Palm OS und 22.3 % Windows CE.

Java™ 2 Platform, Standard Edition (J2SE™) vgl. Homepage von J2SE
normale Java-Version

Java™ 2 Platform, Enterprise Edition (J2RE™) vgl. Homepage von J2RE
Java-Version zur Entwicklung von Unternehmensanwendungen
beinhaltet J2SE, JDBC API, CORBA-Technologie, Support für JavaBeans-Komponenten, Java Servlet API, JavaServer Pages und XML-Technologie

PersonalJava vgl. Homepage von J2RE

Java-Plattform, die eine Virtual Machine und optimierte Java-Klassenbibliotheken für Verbrauchergeräte (insbes. embedded device), die mit dem Netzwerk verbunden werden können oder Standalone sind, zur Verfügung stellt.

Dabei werden nur eine eingeschränkte Anzahl von Klassenbibliotheken verwendet, um vorher feststehende Aufgaben zu erfüllen. Geringerer Memoryverbrauch als für PersonalJava-Anwendungen.

Implementierung: Es gibt eine Runtime Environment für Windows CE.

EmbeddedJava vgl. Homepage von J2RE

Java-Plattform, die eine Virtual Machine und optimierte Java-Klassenbibliotheken für Verbrauchergeräte (Büro, Haushalt, mobile Geräte), die mit dem Netzwerk verbunden werden kann, zur Verfügung stellt.

Implementierung: J2ME

Wichtig: Java ist nicht JavaScript, obwohl die Notation manchmal sehr ähnlich ist. JavaScript gehört eher zu den Skriptsprachen und wird vom Web-Browser interpretiert. Java ist dagegen eine vollständige, sehr umfangreiche, leicht erweiterbare Programmiersprache und nicht auf den Einsatz in Web-Browsern limitiert.

Vorgehen:

Die Java-Programme werden als ganz normale ASCII-Dateien mit irgendeinem Editor (PFE, Notepad, NEdit, XEmacs, ...) unter der Endung ".java" abgespeichert. Diese Java-Programme werden dann compiliert und der resultierende Bytecode (mit der Endung ".class")

interpretiert. Gestartet (und in HTML-Files eingebunden werden also keine Klartext-Dateien, sondern ein sogenanntes binäres File.

wichtige Begriffe:

Java *Programmiersprache*

(amerikan. Slangwort für Kaffee)

Die Programmiersprache wurde zuerst vom Green Project und dann von Sun entwickelt. Die Programmiersprache soll objektorientierte, einfache, portable Programme mit Benutzeroberfläche, Multimedia-Fähigkeiten, ... ermöglichen. Erste Anwendungszwecke: Haushaltsgeräte, Settop-Boxen für Fernsehen, Internet, ...

Applet *Java-Programm*, das zum Start unbedingt einen *javafähigen Web-Browser* benötigt. Der Bytecode des compilierten Java-Programmes wird mit dem APPLET-Befehl in einer HTML-Seite gestartet, wenn diese HTML-Seite vom Web-Browser angezeigt wird.

Applikation *Java-Programm*, das ein *eigenständiges Programm* ist und daher von einer Shell/MSDOS-Box gestartet wird. Normalerweise wird der Bytecode des compilierten Java-Programmes vom Java-Interpreter interpretiert und gestartet. Applikationen haben normalerweise mehr Rechte (weniger Sicherheitseinschränkungen) als Applets, z.B. beim Lesen, Schreiben von lokalen Files, ...

JDK *Java Development Kit*

benötigt man, wenn man Java-Programme *selbst übersetzen* will, zum Start von bereits übersetzten Programmen reicht aus das JRE (von JavaSoft/Sun im Prinzip frei erhältliche *Programmierungsumgebung* inkl. Programmierwerkzeuge wie Appletviewer, Compiler, Interpreter, Debugger, Klassenarchive der Form *.jar und systemabhängige Bibliotheken, ...)

JRE *Java Runtime Environment*

benötigt man, wenn man Java-Programme (Java-Applets und -Applikationen) selbst *starten* will
abgemagertes JDK, das auch von kommerziellen Software-Anbietern zusammen mit deren eigenen Java-Programmen an Kunden ausgeliefert werden darf

JVM *Java Virtual Machine*

muss im Web-Browser enthalten sein, damit Java-Applets starten können
wird durch den Java-Interpreter bereitgestellt, damit Java-Applikationen starten können
genauer: setzt den Bytecode des compilierten Java-Programmes

GUI *Graphical User Interface*

Benutzeroberfläche, gemeint sind die Eingabe- und Ausgabemöglichkeiten von grafischen Programmen (z.B. Web-Browser selbst, Word, StarOffice, ...), die mit Buttons, Anzeige-/Auswahllisten, Menüs, ...)

SDK *Software Development Kit*

Java SDK ist die neue Bezeichnung für JDK

Versionen:

JDK 1.0 völlig veraltet, schlechtes Handling von Windows-Ereignissen wie z.B. Mouseclicks, ...

JDK 1.1 veraltet, hat noch keine neuen modernen GUI-Elemente aber führt Internationalisierung, Sicherheitsaspekte, JavaBeans, Mathematik-Pakete, Aufruf von Programmen auf fremden Rechnern, innere Klassen, ... ein
verbessertes Handling von Windows-Ereignissen
wird *weitgehend* von den allermeisten Browsern (Netscape ab Version 4.0.5, Internet Explorer ab Version unterstützt)

form Version 1.2 relativ neu, enthält jetzt *moderne GUI-Elemente* im Swing-Paket (Buttons mit Bildern, Anzeigelisten, Trees, ...), neue Klassen für 2D-Grafik, Drag & Drop, Unterstützung für behindertengerechte Anzeige (Bildschirmvergrößerer, Spracherkennung, Tastaturunterstützung, ...) und Verbesserungen im Bereich Audio, JavaBeans, JNI, ...
(kaum Unterstützung der neuen Features in Netscape, Internet Explorer, man benötigt ein Plugin)

form Version 1.3 neueste ausgereifte Version, enthält HotSpot Server zur Beschleunigung von Java-Programmen, Unterstützung von Java Sound, Java Naming and Directory Interface, Verbesserungen für Swing, Applets, Sicherheit, Drag & Drop, RMI, 2D-Grafik (kaum Unterstützung der neuen Features in Netscape, Internet Explorer, man benötigt ein Plugin)

form Version 1.4 momentan im Beta-Stadium, soll z.B. XML-Support mit integrieren

0.4 * Geschichte

1975-1980

Bill Joy entwirft Pläne für eine Sprache, die die besten Eigenschaften von MESA und C vereinigt.

1985-1989

Bill Joy und Ingenieure von Sun starten Revision des UNIX-Betriebssystems (Verbindung von SunOS 4.x und SYS V R4 von AT & T)

1989

März Tim Berners-Lee (CERN, Genf) veröffentlicht 1. Projektvorschlag für Hypertext-Oberfläche (Ziele: einfach zu bedienen, systemübergreifend)

- Bill Joy verläßt Sun und investiert in Microsoft.

1990

- Bill Joy veröffentlicht den Artikel “Further“ (Vorschlag an Sun für eine objekt-orientierte Umgebung basierend auf C++)
- James Gosling (emacs) arbeitet an einem SGML-Editor “Imagination“ und benutzt C++. Frustriert von C++ entwickelt er die Sprache *Oak*.

5. Dez Patrick Naughton startet das *Green Project* mit Mike Sheridan und James Gosling (Leitung). Der Artikel von Bill Joy, der gerade an Grafik und User Interfaces in C arbeitet, beeinflusst das Team.

1991

April Das Green Project verlegt die Anwendung auf hybride Systeme im Consumer Electronic-Bereich (Ziel: interaktive Steuerung von Waschmaschinen, Telefonanlagen, ...). James Gosling entwickelt Oak verstärkt weiter (Compilerentwicklung in C, Runtime-Interpreter in C).

August erste Oak-Programme laufen
(Ziel von Oak: Hochsprache für Consumer Electronic-Bereich, einfacher als C++)

1992

- 1. Prototyp läuft auf “*7“ (Mischung aus PDA und Fernsteuerung) mit grafischer Benutzeroberfläche und animierten Figuren (Tumbling Duke, das Maskottchen von Java)
- Aus dem Green Project wird die Firma *First Person* (Tochter von Sun).

1993

- Time-Warner schlägt eine Settop-Box zur Realisierung vor mit einem OS, First Person bewirbt sich und ändert die Forschungsrichtung auf interaktives TV, Video on Demand, Eine Zusammenarbeit entsteht jedoch nicht, das Projekt wird von Time-Warner abgelehnt.

Frühling ca. 50 WWW-Server weltweit

1. Version von *Mosaic* (WWW-Browser) von NCSA

Die WWW-Seiten verwenden HTML (Hypertext Markup Language) und GIF-Bilder. Zur Kommunikation wird HTTP (Hypertext Transfer Protocol) eingesetzt.

Oktober ca. 200 WWW-Server weltweit

(1 % des Datenverkehrs auf dem NSF-Backbone sind bereits WWW-bedingt)

1994

- First Person wird aufgelöst, das Team wieder in Sun eingegliedert.
- Green Team erkennt ähnliche Grundvoraussetzungen an die Programmiersprache bei Settop-Boxen, Consumer Electronics und Internet-Programmen (Sicherheit, architekturneutrale Codeausführung auf fremden Rechnern (untrusted hosts) mit verschiedenem OS)

- Patrick Naughton entwickelt den Prototyp “WebRunner“ für WWW-Browser an einem Wochenende. Zusammen mit Jonathan Payne wird daraus später der *Hot-Java*-Browser, der auf Java-Konzepten beruht und dynamischen Web-Inhalt verarbeiten kann (alpha3-Release, kein Sicherheitskonzept, kein Applet-API)

März Marc Andreessen verläßt mit Kollegen NCSA, um mit Jim Clark (SGI) die Firma Mosaic Communications Corp. (später Netscape Communications Corp.) zu gründen, um Mosaic und andere WWW-Technologien zu vermarkten.

Oktober 1. Beta-Version des *Netscape*-Browsers

- Der Name *Java* (Slang-Wort für Kaffee als aufmunterndes Gebräu) entsteht in Ermangelung eines besseren Vorschlags.
- Einbindung von kleineren Java-Programmen (Applets) durch spezielle HTML-Tags in WWW-Seiten geplant (→ dynamische Seiten)

1995

Mai Weiterentwicklung von HotJava, der der 1. Browser ist, der Applets unterstützt und selbst die 1. größere Java-Applikation ist
 “hotjava“ ist selbst Java-Klasse, enthält den C-Source für JVM sowie grundlegende Java-Klassen (JVM = Java Virtual Machine).

23. Mai Sun stellt Java auf SunWorld in San Francisco vor

- James Gosling und Henry McGilton veröffentlichen White Paper “The Java Language Environment“ bei Sun
- Sun veröffentlicht “Java API Users Guide“ (API = Application Programming Interface)

ab 31. Aug Verschiedene Firmen lizensieren Java:

31. August '95:	Netscape (oder bereits im Mai?)
30. Oktober '95:	Oracle
02. November '95:	Borland
01. Dezember '95:	SGI
06. Dezember '95:	Adobe und IBM
13. Dezember '95:	Symantec
12. März '96:	Microsoft
21. März '96:	Novell
26. Juni '96:	Corel Corp.
27. März '97:	Siemens

Java wird in die Version Netscape 2.0 eingebaut.

Juli-Oktober 1. Beta-Version des JDK erscheint (JDK = Java Development Kit)

04. Dez Netscape und Sun kündigen *Javascript* an.

1996

- Es erscheint

James Gosling, Bill Joy, Guy Steele: The Java language specification.

in Sun Press, die Sprachdefinition von JDK 1.0.

- erste IDEs für Java (Borland, IBM, Microsoft, Sybase/Watcom, Symantec, ...)
- HotJava 1.0 erscheint mit modularer Architektur (nachladbare Module, multi-threaded, HTML 2.0+, Applets, Klassenlibraries für die Erstellung von Web-Applikationen inkl. Browser-Klasse mit HTML-Parser, Web-Navigation, ...)

Januar JavaSoft veröffentlicht JDK 1.0.

20. Feb Sun kündigt JDBC API an

30. Apr Sun kündigt an, daß die Firmen

Apple, HP, IBM, Microsoft, Novell, SCO, SGI

die Java-Plattform in ihr OS einbetten wollen.

17. Mai Sun lizenziert JavaOS und HotJava-Browser (am 6. November auch IBM)

29. Mai JAVAONE (1. Worldwide Developer Conference in San Francisco)
Ankündigung der Java APIs

Media (2D, 3D, Animation, ...), RMI, Security, Servlet

und von JavaOS

15. Sep Das Buch

The JVM Specification and the Java Class Libraries

erscheint bei Addison & Wesley.

14. Okt Corel Office for Java wird ausgeliefert.

26. Okt Sun kündigt JIT-Compiler an (JIT = Just in Time)

01. Dez Sun verkauft 1. JavaStation

03. Dez Sun kündigt JDK 1.1 an

11. Dez Sun startet die "100 % Pure Java"-Initiative auf der "Internet World" in New York

1997

11. Jan 1. Version von Java Beans Development Kit

18. Feb Sun veröffentlicht JDK 1.1

26. Feb StarDivision veröffentlicht "StarOffice for Java", Vorstellung auf der JAVAONE-Konferenz

05. März JavaOS wird ausgeliefert.

11. März Sun liefert JRE aus (JRE = Java Runtime Environment)

17. März Sun Microsystems Inc. bittet das Standardisierungskomitee ISO/IEC JTC1 (IEC = International Electrotechnical Commission, JTC = Joint Technical Committee) um den Status als PAS-Spezifizierer (PAS = Publicly Available Specification) für Java. Damit könnte Sun direkt Standardisierungsvorschläge an ISO/IEC schicken und so die Standardisierung beschleunigen.

28. März Sun veröffentlicht JDK 1.1.1

- 02. Apr** JAVAONE in San Francisco
 - Netscape und Sun planen JFC (Java foundation classes)
 - Sun lizenziert Symantec JIT-Compiler
- 02. Juni** Sun veröffentlicht JDK 1.1.2.
- 17. Juni** Sun veröffentlicht JDK 1.1.3.
- 08. Juli** Netscape und Sun liefern JFC Developer Release aus
- 16. Juli** ISO/IEC JTC 1 stimmt erstmalig ab und lehnt zunächst das Ansinnen von Sun auf PAS-Spezifiziererstatus ab (3 Ja-Stimmen, 5 Ja-Stimmen mit Kommentaren, 15 Nein-Stimmen mit Kommentaren, 1 Enthaltung) Nach den Statuten hat Sun jetzt 60 Tage Zeit, um auf die Kommentare zu antworten, eine endgültige Ablehnung ist diese Abstimmung noch nicht.
- 22. Sep** Sun antwortet auf die Kommentare von ISO/IEC JTC 1.
- 07. Okt** Sun verklagt Microsoft, da der Internet Explorer 4.0 und SDK for Java 2.0 als Java 1.1-kompatibel angekündigt waren, obwohl sie die Java 1.1-Testsuite nicht erfolgreich durchlaufen haben. Es fehlen die Packages JNI und RMI, und es sind interne Java Core-Klassen von Microsoft implementiert worden, die durch je 50 neue Methoden und Felder verändert worden sind. Dies würde natürlich die Portabilität von Java-Code gefährden.
- 17. Nov** ISO/IEC JTC 1 stimmt zum 2. Mal ab und genehmigt Sun den PAS-Spezifiziererstatus (20 Ja-Stimmen, 2 Nein-Stimmen der USA und China, 2 Enthaltungen). Die Abstimmung in den USA fand im Microsoft Headquarter in Redmond statt und erbrachte statt der 2/3-Mehrheit nur 60 % Zustimmung.
- 11. Dez** Java Activator Early Access Beta 1 wird auf der Internet World Fall vorgestellt (Plugin für Microsoft Internet Explorer 3.0 und später bzw. Netscape 3.02 und später, das eine JDK 1.1-kompatible JVM liefert). Die HTML-Seiten müssen allerdings umgewandelt werden, bevor das Plugin aktiv und die browsereigene JVM umgangen wird.
- 18. Dez** JDK 1.2 Beta 2 wird veröffentlicht.

1998

- 24. Jan** Java Activator Early Access Beta 2 wird veröffentlicht
- 26. Feb** JFC 1.1 (Swing 1.0) wird veröffentlicht
- 03. März** Java Activator Early Access Beta 3 wird veröffentlicht
- 17. April** Netscape Navigator 4.05 enthält Java 1.1-Support (kein gesonderter Patch mehr erforderlich). Allerdings schmückt sich der Navigator nicht mit dem Java-Logo, da eine 100 %-ige Kompatibilität nicht besteht.
- 26. April** JDK 1.1.6 wird veröffentlicht
 - 13. Juli** Java Plugin 1.1.1 (früher Java Activator) wird veröffentlicht
 - 16. Juli** Kaffe 1.0 (frei JVM) wird veröffentlicht
 - 22. Juli** JDK 1.2 Beta 4 wird veröffentlicht
- 21. Aug** Der 1. Java-Virus wird bekannt. Er wird jedoch erst wirksam, wenn er im Appletviewer oder mit lokaler JRE gestartet wird.

- 29. Sep** JDK 1.1.7 wird veröffentlicht
- 08. Okt** Nachdem Intel und SGI die Java-Multimedia-Gruppe verlassen haben, wollen Sun und IBM die JMD 2.0 entwickeln.
- 23. Okt** JDK 1.1.7A wird veröffentlicht
- 03. Nov** Sun unterstützt die Java-Portierung auf Linux.
- 04. Nov** Die Real-Time Java Working Group wird mit 15 Mitgliedern (u.a. Siemens, HP, Microsoft) gegründet. Sie fordert einen offenen Standardisierungsprozeß für Java, Sun wird eingeladen, in der Gruppe mitzuwirken.
- 18. Nov** Microsoft muß nach einer richterlichen Verfügung, innerhalb von 90 Tagen den Java-Code ihres Produktes modifizieren, so daß es den Java-Standard entspricht.
- 20. Nov** JINI (Java Intelligent Architecture) Software Kit (JSDK) wird veröffentlicht
- 04. Dez** JDK 1.2 für Windows 32 wird veröffentlicht (enthält JDBC 2.0, Java-IDL mit CORBA, Jahr 2000-sicher). Die Lizenzbestimmungen wurden geändert, der Sourcecode darf jetzt für eigene Produkte modifiziert werden, er darf erweitert werden, ohne daß man diese Änderungen an Sun weiterleiten muß.
- 17. Dez** Microsoft widerspricht gerichtlich der einstweiligen Verfügung, die Auslieferung von Windows 98 und Visual J++ 6.0 zu stoppen.

1999

- 25. Jan** 1. JINI-Präsentation in San Francisco, bei der ein JINI-fähiges Gerät über das Netz seine Dienste anbietet. Bosch plant einen JINI-fähigen Geschirrspüler.
- 09. Feb** Microsoft reicht bei Gericht die Frage ein, ob sie eine eigene Java-Version entwickeln dürfen.
- 15. Feb** Microsoft bekommt von einem US-Bezirksrichter die Erlaubnis, eine eigene Java-Version entwickeln dürfen. Eine unabhängig entwickelte "clean room"-Entwicklung würde das Copyright von Sun nicht verletzen.
- 20. März** Bei der Cebit-Messe wird eine JINI-fähige Kaffeemaschine vorgeführt.
- 30. April** Die HotSpot-Performance Engine wird von Sun zum kostenlosen Download angeboten. Diese Technologie soll den Ablauf der Java-Programme beträchtlich steigern.

2000

- 20. Feb** Forte for Java, Suns freie IDE für Java, erscheint in einer Beta-Version für Solaris, Linux, Windows 98 und Windows NT
- 21. Mär** Java 2 Platform, Standard Edition (J2SE) v 1.2.2 für Linux erscheint
- 18. Apr** Forte for Java, Suns freie IDE für Java, erscheint in Version 1.0 für Solaris, Linux und Windows-Plattformen
- 08. Mai** Java 2 Platform, Standard Edition (J2SE) v 1.3 für Windows 95, Windows 98, Windows NT 4.0 und Windows 2000 sowie Beta-Versionen für Solaris und Linux erscheinen
- 25. Mai** Java 2 Platform, Enterprise Edition (J2SE) v 1.2.1 erscheint und unterstützt Solaris 7 und 8, Linux and Windows NT und Windows 2000

- 31. Mai** Oracle lizenziert Java 2 Platform Enterprise Edition (J2EE)
- 06. Jun** Java 2 Platform, Standard Edition (J2SE) v 1.3 beta und Java 2 Platform, Standard Edition (J2SE) v 1.2.1 für Linux erscheinen
- 12. Sep** Java Cryptography Extension (JCE) v1.2.1 und Java Secure Socket Extension (JSSE) v1.0.2 erscheinen
- 27. Sep** Java 2 Platform, Standard Edition (J2SE) v 1.3 für Solaris erscheint
- 04. Okt** Java 2 Platform, Standard Edition (J2SE) v 1.3 für Linux erscheint (Zusammenarbeit mit Blackdown Team)
Java 2 Platform, Enterprise Edition v1.2.1 unterstützt jetzt auch J2SE v 1.3
- 18. Okt** Jini Technology Starter Kit Version 1.1 erscheint und enthält Klassen zur Programmierunterstützung aufbauend auf die Jini Technology Core Platform
Sun schließt sich einer neuen Industriellianz an, der Internet Home Alliance, siehe Homepage der Internet Home Alliance
- 06. Dez** Java XML API erscheint (XML = Extensible Markup Language) mit 3 neuen Java APIs:
Java API for XML Messaging (JAXM)
Java API for XML Parsing (JAXP)
Java API for XML Data Binding (JAXB)
Sun kündigt enge Zusammenarbeit mit Apache Software Foundation auf drei Schwerpunkten an:
API für Scalable Vector Graphics (SVG)
Jakarta: Projekt rund um JavaServer Pages- und Java Servlets-Technologie
Xerces 2: XML Parser
vgl. Homepage von Tomcat,
Homepage von XML-Support von Apache
- 20. Dez** Java Web Start v 1.0 erscheint
Mit Java Web Start können Java-Anwendungen durch Anklicken auf einen Link in einem Webbrowser geladen und gestartet werden. Dabei braucht der Webbrowser gar nicht Java-fähig sein, es werden automatisch benötigter Java-Support heruntergeladen. Basiert auf Java Network Launcher Protocol (JNLP).

2001

- 20. Jan** Microsoft und Sun beenden den Rechtsstreit um Java, Microsoft zahlt 20 Millionen Dollar an Sun wegen unrechtmäßiger Nutzung des Java Kompatibilitätslogos und akzeptiert Suns Kündigung der Java-Lizenz. Sun erlaubt den Weitervertrieb von Microsofts alten Java-Versionen, solange neue Versionen tatsächlich Java-konform sind.
vgl. Pressemitteilung von Sun
- 11. Apr** Java 2 Platform Micro Edition (J2ME) erscheint für Linux und VxWorks
Unter Linux wird Connected Device Configuration (CDC) und Connected Limited Device Configuration (CLDC) unterstützt. CDC (2 MB oder mehr) wird unterstützt von digitalen TV Setup-Boxen, Bildschirmtelefone, ...
CLDC (512 K-2 MB) in Handys, Pagers, ...
vgl. Homepage von J2ME

- 29. Mai** Beta-Release von Java 2 Platform, Standard Edition v 1.4 erscheint
- 04. Jun** Sun kündigt an, dass Extensible Hypertext Markup Language (xHTML) Microbrowser-Komponenten kostenlos entwickelt werden für alle drahtlosen Geräte, die Java 2 Platform, Micro Edition (J2ME™) unterstützen. Die Microbrowser-Komponenten sollen die Entwicklung von Java-Browsern und -Programmen erleichtern, die den WAP 2.0-Standard unterstützen (WAP = Wireless Application Protocol).
Forte for Java, Suns IDE für Java, Version 3.0 erscheint für Solaris, Linux, Windows
- 16. Jul** Java 2 Platform, Enterprise Edition (J2EE™) Version 1.3 beta 2 erscheint

0.5 * Ziele

Das White Paper von J. Gosling und H. McGilton (1995) definiert als Ziele:

„Java: Eine einfache, objektorientierte, verteilte, interpretierte, robuste, sichere, architektur-neutrale, portable, dynamische und parallele Sprache mit hoher Performance.“

a) **einfach**

- verständlicher als C++ oder Eiffel
- ähnliche Sprachelemente wie in C/C++ (manchmal jedoch mit anderer Bedeutung)
- alle Datentypen außer den einfachen (ganzzahlige, Fließkomma, ...) sind Klassen
Für die einfachen Datentypen gibt es Wrapper-Klassen.
- Streichen „überflüssiger“ C-Konstrukte wie z.B. `struct`, `union`, `enum`, `goto`
- Verzicht auf Zeiger und Zeigerarithmetik (sehr fehlerträchtig!) Objekte werden automatisch durch implizite Referenzen verwaltet und durch „`new`“ neu angelegt.
- automatische Speicherverwaltung (garbage collection) durch eigenen Thread (Prozeß)
Dabei wird der Speicherplatz für nicht mehr benötigte Objekte (solche, auf die keine Variable mehr verweist) wieder automatisch freigegeben.
- umfangreiche Klassenbibliotheken (z.B. Klassen für Datum, dynamischer Vektor, URL, Stack, Zufallsgenerator, ...)

b) **objektorientiert (OO)**

- Schwerpunkt der OO-Programmierung liegt auf den Daten (→ Objekte) und deren Interaktionen (→ Methoden), nicht auf der hierarchischen Strukturierung des Problems in Funktionen/Prozeduren
Kernstück: Klassenkonzept (Vereinigung von Daten und Methoden eines Datentyps)
- vollständig objektorientiert (nur einfache Datentypen sind selbst keine Klassen; es existieren aber Wrapper-Klassen für sie)
Selbst Java-Programme sind Klassen!
- Java unterstützt

Datenkapselung:	Sichtbarkeit von Daten, Verstecken der Implementierungsdetails (data hiding)
Polymorphismus:	verschiedene Funktionen können denselben Namen besitzen, Auswahl der passenden Instanz erfolgt durch die Anzahl/Typ der Argumente)
Vererbung:	Klassen erben Daten/Methoden von anderen Klassen; es entsteht durch „is a“-Verknüpfung eine Klassenhierarchie

- sinnverwandte Klassen bilden ein Package (Paket)

c) **verteilt/kommunizierend**

- Java unterstützt mit high-level-Funktionen Netzwerk-Kommunikation über TCP/IP. Wichtige Protokolle wie HTTP, FTP, ... sowie Klassen für URLs, URL-Verbindungen sind Bestandteile von Java. Es gibt außerdem Packages für RMI (RMI = Remote Method Invocation) und Servlets (kommunizierende Java-Programme, die CGI-Programme ersetzen sollen).

Low-level-Funktionen unterstützen die Kommunikation mit Datagrammen (UDP) und Sockets für stream-basierte Verbindungen.

Allein der Start von Applets in WWW-Browsern setzt eine Netzwerkkommunikation voraus.

d) **interpretiert**

Der Java-Compiler „`javac`“ erzeugt nicht Maschinencode, sondern einen Bytecode. Zum Ausführen eines Java-Programms muß daher der Java-Interpreter „`java`“ gestartet werden, der den kompilierten Bytecode ausführt. Der Bytecode ist ein architekturneutrales Format für den Objectcode und kann problemlos zwischen verschiedenen Rechner unterschiedlichen Betriebssystemen ausgetauscht werden.

Beim Java-Compiler gibt es also keine Linkphase (wenn man vom Nachladen von Java-Klassen absieht), die Programmherstellung und Testphase ist bei Java also verkürzt. Allerdings ist, durch den Interpreter bedingt, die Laufzeit eines Java-Programms gegenüber einem C-Programm bis zu 20 Mal langsamer. Modernere JIT-Compiler versuchen, bei mehrmaligem Aufruf gleicher Methoden den beim ersten Mal erstellten Maschinencode für die weiteren Aufrufe zu nutzen. Executables, die von Java-Compiler stammen, die tatsächlich Maschinencode erzeugen, sollen vergleichbar schnell wie C-Executables sein.

Der Interpreter und die Runtime-Umgebung bilden die *JVM* (Java Virtual Machine). Da Applets (spezielle Java-Programme) einer WWW-Seite von dem Web-Browser verarbeitet werden müssen, muß in den Browsern eine JVM eingebaut sein, die den Bytecode umsetzt in das Betriebssystem des vorliegenden Rechners. Die Idee der Bytecode-Erstellung ist keinesfalls neu und wurde bereits bei dem Übersetzen von UCSD-Pascal in P-Code genutzt.

e) **robust**

Während viele PC-Benutzer oft und meist gelassen ihren PC rebooten bei einem Absturz, ist dies im Bereich der Consumer Electronic nicht erwünscht.

Java arbeitet besonders robust aufgrund der folgenden Eigenschaften:

- strenge Typisierung (automatische Typumwandlungen gibt es nicht, Typumwandlungen müssen durch explizite Casts erfolgen, implizite Methodendeklarationen wie in C sind nicht erlaubt)
- automatische Speicherverwaltung und der Verzicht auf Zeiger (Vermeidung von Speicherlecks, schwer zu findenden Zeigerfehlern)
Durchschnittlich sind in 55 C-Codezeilen ein Bug enthalten, wobei ungefähr die Hälfte der Fehler auf falsche Verwendung von Zeigern zurückgehen.
- Überprüfung von Objekt-, String- und Arrayzugriffen zur Laufzeit, so daß unzulässige Zugriffe abgefangen werden
- Ausnahmekonzept (exception handling) zur einheitlichen Behandlung von Fehlern und Ausnahmesituationen
Der Fehler/die Ausnahme muß nicht am Ort des Entstehens behandelt werden, dies kann an einer zentralen Stelle erfolgen.

f) **sicher**

Absolute Sicherheit kann wohl nie erreicht werden, doch Hacker werden bei Java mit verschiedenen Sicherheitsmechanismen konfrontiert. Da eine Ausführung von Programmen, die vom Internet geladen werden, stets Sicherheitsbedenken (Virengefahr, unberechtigte Zugriffe auf lokale Festplatten) nach sich zieht, legt Java einen hohen Wert auf Sicherheitsmechanismen.

Es kann nicht verhindert werden, daß Applets bei der Ausführung eine Unmenge von Memory anfordert, viele Threads startet oder sehr viele Daten über das Netz nachlädt. Dadurch kann das lokale System erheblich verlangsamen (“denial of service attack“).

- Java-Interpreter enthalten Bytecode-Verifizierern (Prüfung auf gültigen JVM-Code, Overflow/Underflow des Stacks, inkorrekte Registerzugriffe, illegale Typkonvertierungen).
- Java-Interpreter verstehen Authentifizierungs- und Zertifizierungsprotokolle (mit kryptografischen Verfahren). Applets können mit einer digitalen Unterschrift versehen werden. Solche Applets von trusted hosts können von anderen Usern als vertrauenswürdig eingestuft werden und weitergehende Zugriffsmöglichkeiten erhalten.
- JVMs in WWW-Browsern ist es normalerweise nicht erlaubt, auf das lokale Dateisystem zuzugreifen. Andere Netzverbindungen als die zu dem Host, der das Applet bereitstellt, sind normalerweise nicht erlaubt. Es gibt verschiedene Sicherheitsstufen für den Appletviewer vom JDK, untrusted applets, trusted applets und Java-Applikationen.
- Für Java-Applikationen kann ein Security-Manager gestartet werden, der Datei- und Netzzugriffe unterbinden kann.
Für Applets wird automatisch ein Security-Manager gestartet, der nicht ersetzt werden kann.
- Der Verzicht auf Zeiger ist zudem eine Sicherheitsmaßnahme, da Zeiger und Zeigerarithmetik stets dazu mißbraucht werden können, Speicherbereiche außerhalb des Programmes zu adressieren oder zu modifizieren. Der Java-Compiler verzichtet auf Memorylayout, so daß der Bytecode keinerlei Hinweise auf spätere tatsächliche Memoryadressen liefert.

- Separate Namespaces für über das Netz geladene Klassen und für lokale Klassen verhindern, daß die vom Netz geladenen Klassen die Standard-Java-Klassen ersetzen können.
- Untrusted Applets können nur auf wenige Systemeigenschaften zugreifen (OS-Version, OS-Name, File- und Pfadtrennzeichen, ...), nicht jedoch auf den Benutzernamen und -verzeichnis, das Installationsverzeichnis der Java-Umgebung,
- Untrusted Applets können keine neuen Prozesse starten, den Java-Interpreter verlassen oder dynamische Libraries des lokalen Systems nachladen.

g) **architektur-neutral**

- Java-Compiler erzeugen einen Bytecode (wie oben beschrieben), der erst von der JVM in einen systemabhängigen Maschinencode umgewandelt wird. Damit kann jedes Java-Programm ausgeführt werden, wenn das System eine JVM bereitstellt (z.B. mit dem JDK oder innerhalb eines Web-Browsers).
Die JVM besitzt eine relativ kleine Menge von Instruktionen, die sich gut auf heutige 32-Bit-Prozessoren umsetzen lassen.
- Der Java-Compiler ist selbst in Java geschrieben, während das Java Runtime-System in ANSI C geschrieben wurde und ist im wesentlichen POSIX-kompatibel. Damit kann die Java-Umgebung selbst auf neue Betriebssysteme portiert werden.
- Die Benutzung des AWT (AWT = Abstract Window Toolkit) stellt sicher, daß Java-Programme das passende Erscheinungsbild und Verhalten auf verschiedenen Systemen haben.

h) **portabel**

- Architektur-Neutralität ist eine wesentliche Grundvoraussetzung für Portabilität. Bisher gibt es Portierung auf Solaris, Windows 95, Windows NT, IRIX 5.3 und 6.x, HP-UX 10.x, Linux und sogar auf Windows 3.1 (IBM).
- Java spezifiziert die Größe von primitiven Datentypen (wie `int` und `double`).
- Java unterstützt durch Verwendung des Unicode-Zeichencodes internationale Sonderzeichen (16-Bit-Zeichen, echte Obermenge des ASCII-Zeichensatzes). Damit kann man auch nationale Sonderzeichen für Bezeichner und in Strings einsetzen.
- Java unterstützt in vielen Bereichen die Internationalisierung, d.h. viele Klassen unterstützen eine länderspezifische Ausführung durch Laden von Lokalisierungsinformationen. Als Beispiele können hier die Datums- und Zeitausgabe, Zahlen- und Währungsausgabe, Sortierung von Strings (Berücksichtigung von nationalen Sonderzeichen), lokale Zeichenkodierung (I/O-Textausgabe durch Umwandlung von Unicode in lokalen Zeichensatz und umgekehrt) genannt werden. Java stellt dazu eine *Locale*-Klasse bereit. Eine *Locale* ist definiert als politische, geographische oder kulturelle Region, die eine bestimmte Sprache oder Konventionen besitzt zur Darstellung von Datum, Zeit, Zahlen, Jedes Java-Programm stellt zunächst die Default-Locale (des Betriebssystems) bereit. Java-Programme können sog. Locale-abhängige Resource-Bundles bereitstellen, z.B. für die Bezeichnung von Labels, Programmierungen,

i) **dynamisch**

- Java erlaubt das dynamische Nachladen von Klassen, die dynamisch instanziiert werden können. Dies ist insbesondere wichtig für die Verteilung von Software über das Internet (→ HotJava-Browser).
- Jede Java-Klasse kann dynamisch Informationen über sich selbst zur Laufzeit zur Verfügung stellen.
- In Java ist es trotz fehlender Zeiger leicht möglich, die Länge von Arrays dynamisch zur Laufzeit zu verändern (`new`-Operator) und dynamisch verkettete Datenstrukturen wie (Stacks, Queues, Bäume, ...) zu erzeugen. Java stellt einige solcher dynamischen Datenstrukturen bereits zur Verfügung (dynamischer Vektor, Stack, Hashtabellen, Bitsets, ...).

j) **parallel/multithreaded**

- Java unterstützt die Parallelisierung von Programmen durch die `Thread`-Klasse (Thread = Ausführungsfaden), auch wenn diese Prozesse zunächst nur auf einem Prozessor ausgeführt werden. Ein Thread ist ein separater Prozeß mit eigenem Programmzähler und Stack, aber mit Zugriff auf alle Objekte des Java-Programms.
- Die Synchronisierung sowie das kurzzeitige Anhalten, das Weiteraufnehmen und Stoppen von Threads ist in Java als Sprachbestandteil vorhanden.
- Der Java-Interpreter startet selbst automatisch Threads (z.B. je einen für die garbage collection, das Bilderladen, Benutzereingaben mit Maus/Tastatur, Neuzeichnen von Applets ...) bei der Abarbeitung eines Java-Programms.

k) **schnell**

Dies kann nicht uneingeschränkt ohne weitere Bemerkungen behauptet werden. Die Geschwindigkeit ist eines der größten Hinderungsgründe für Software-Hersteller, Java in ihren Programmen einzusetzen.

- Java ist im Vergleich zu anderen OO-Programmiersprachen wie CLOS oder Smalltalk schneller.
- Java ist im Vergleich zu anderen Interpretersprachen wie Perl, Tcl/Tk oder Unix-Shells schneller.
- Java ist um den Faktor 10 bis 20 langsamer als kompilierte C-Programme. JDK 1.1 ist etwa doppelt so schnell wie JDK 1.0. Einen teilweisen Ausweg bieten JIT-Compiler, die zumindest partiell Code aus Maschinenbefehlen erzeugen. Sun behauptet, daß die Performance damit mit C- oder C++-Programmen vergleichbar sei.
- Typische Anwendungen für Java-Programme wie Frontends für Datenbanken, C-Programme, ... oder netzwerkbasierte Applikationen benötigen keine hohe Geschwindigkeit, da sie die meiste Zeit auf Benutzereingaben oder auf Datenempfang warten. In JDK 1.1 wurde zudem das Event-Handling effizienter gestaltet.

0.6 * Java-Applikationen und -Applets

Wie schreibe und starte ich eine Java-Applikation?

- 0) Falls nötig, installiere JDK Version 1.1 oder 1.2 (Java 2 Platform).
Das Verzeichnis mit `appletviewer.exe`, `javac.exe`, ... (bzw. unter Linux `appletviewer`, `javac`, ...)

<Installationsverzeichnis vom JDK>\bin

muß im Pfad liegen (Windows 95/98 in `autoexec.bat`), siehe Dokumentation zur Installation!

- 1) Erstelle mit einem Texteditor ein Textfile namens "Bsp_01.java" mit der Dateieindung ".java", z.B. im Verzeichnis "C:\work" (Linux-User: "\${HOME}/work")

```
public class Bsp_01
{
    public static void main(String [] argv)
    {
        System.out.println("Java ist einfach schoen!");
    }
}
```

- 2) Windows 95/NT: Start einer DOS-Box (Start → Programme → MS-DOS-Eingabeaufforderung)
Linux: Start eines Terminalfensters (abhängig vom Windowmanager bzw. starte "xterm &")

3) Windows 95/NT: Wechsle in der DOS-Box in das richtige Verzeichnis:

```
cd C:\work
```

Linux: Wechsle in dem Terminalfenster in das richtige Verzeichnis:

```
cd $HOME/work
```

S 81: Start des Batch-Files (enthält DOS-Kommandos) für JDK-Version 1.1.4

```
k:\start
```

bzw. von JDK-Version 1.2.1

```
k:\start_121
```

4) Starte in der DOS-Box/dem Terminalfenster den Java-Compiler "javac":

```
javac Bsp_01.java
```

Es entsteht das File "Bsp_01.class", das den portablen Bytecode enthält.

5) Starte in der DOS-Box/dem Terminalfenster den Java-Interpreter "java" mit dem Namen der öffentlichen Klasse:

```
java Bsp_01
```

Der Java-Interpreter startet die Interpretation des Bytecodes des Files "Bsp_01.class" für die Klasse "Bsp_01". Damit startet das Java-Programm.

mögliche Fehlerquellen beim Schreiben/Starten von Applikationen:

- Die öffentliche Klasse heißt nicht genauso wie das File (ohne die Endung ".java").
- Bei der Klassendefinition wurde "public" vergessen.
- Die erste Zeile der "main()"-Funktion wurde nicht genauso wie im Beispiel spezifiziert.
- Beim Start des Java-Interpreters wurde versehentlich die Endung ".class" der Klasse angegeben, d.h. er wurde mit dem Befehl

```
java Bsp_01.class
```

gestartet.

- Das File ".class" wurde als Executable (ausführbares Programm) angesehen und ohne den Java-Interpreter gestartet:

Bsp_01.class

- Das Java-Programm ist ein Java-Applet und gar keine Java-Applikation.

Erläuterungen (Details):

Klasse	eigener Datentyp mit Datenelementen und Funktionen
Objekt/Instanz	Variable vom Datentyp einer Klasse (besitzt objektabhängige (und ggf. klassenabhängige) Datenelemente und Funktionen als Komponenten)
Methode	Funktion, die eine Klasse bereitstellt
Paket	Sammlung sinnverwandter Klassen (Packages)
Instantiierung	Anlegen eines Objekts einer Klasse
<code>class</code>	“Bsp_01“ ist Name einer Klasse, deren Klassendefinition im File “Bsp_01.java“ steht
<code>public</code>	Die Klasse “Bsp_01“ bzw. die Methode “main()“ ist öffentlich zugänglich, d.h. es können Objekte außerhalb der Klassendefinition instantiiert werden bzw. die Methode kann außerhalb der Klassendefinition aufgerufen werden
<code>static</code>	Modifizierungsattribut, das angibt, das das Element/die Methode klassenspezifisch ist. Damit teilen sich alle Objekte der Klasse “Bsp_01“ genau <i>eine</i> Instanz der “main()“-Methode. Ohne den Zusatz “static“ sind die Elemente/Methoden objektabhängig, d.h. jedes Objekt der Klasse hat sein eigenes Element/seine eigene Methode.
<code>void</code>	signalisiert, daß die Methode “main()“ keine Rückgabe bereitstellt (leerer Datentyp)
<code>String</code>	im Package “java.lang“ definierte Klasse, die Java für Zeichenkette bereitstellt
<code>argv []</code>	Vektor von Elementen vom Typ <code>String</code> , der die an “main()“ übergebenen Argumente enthält <code>argv[0]</code> : 1. Argument als <code>String</code> <code>argv[argv.length-1]</code> : letztes Argument als <code>String</code> Wird kein Element übergeben, entspricht <code>argv</code> einem Vektor der Länge 0 und enthält keine zugreifbaren Vektorkomponenten.

System	im Paket "java.lang" vordefinierte Klasse für die Systemumgebung, die u.a. das statische/klassenspezifische Element "out" besitzt
System.out	Zugriff auf das statische Element "out" der Klasse System, das ein Objekt der Klasse "java.io.PrintStream" ist und u.a. eine Methode "println()" besitzt
System.out.println()	Aufruf der Methode <div style="text-align: center;">void println(String s)</div> des Klassenobjekts "System.out", die einen String auf der Standardausgabe ausgibt und danach eine neue Zeile beginnt
"Java ist einfach schoen!"	Objekt der Klasse String, das die in Doppelanführungszeichen eingeschlossene Zeichenkette enthält
";"	dient zum Abschluß von Anweisungen wie in C/C++ Damit muß jede Anweisung (außer der Blockanweisung) mit ";" beendet werden.
."	dient zur Referenzierung von Elementen/Methoden von Klassen/Objekten.
"{...}"	Blockanweisung (eine/mehrere Anweisungen werden zu einer Anweisung zusammengefaßt) Wird im Beispiel zur Definition des Funktionsblockes in "main()" bzw. zur Gruppierung aller Klassenelemente und -methoden von "Bsp_01" verwendet.

Wie schreibe und starte ich eine Java-Applet?

- 0) Falls nötig, installiere JDK Version 1.1 oder 1.2 (Java 2 Platform).
Das Verzeichnis mit appletviewer.exe, javac.exe, ... (bzw. unter Linux appletviewer, javac, ...)

<Installationsverzeichnis vom JDK>\bin

muß im Pfad liegen (Windows 95/98 in autoexec.bat), siehe Dokumentation zur Installation!

- 1) Erstelle mit einem Texteditor ein Textfile namens "Bsp_02.java" mit der Dateieindung ".java", z.B. im Verzeichnis "C:\work" (Linux-User: "\${HOME}/work")

```
public class Bsp_02 extends java.applet.Applet
{
    public void paint(java.awt.Graphics g)
    {
        g.drawString("Java startet als Applet!", 20, 50);
    }
}
```

- 2) Windows 95/NT: Start einer DOS-Box (Start → Programme → MS-DOS-Eingabeaufforderung)
Linux: Start eines Terminalfensters (abhängig vom Windowmanager bzw. starte "xterm &")

- 3) Windows 95/NT: Wechsle in der DOS-Box in das richtige Verzeichnis:

```
cd C:\work
```

Linux: Wechsle in dem Terminalfenster in das richtige Verzeichnis:

```
cd $HOME/work
```

S 81: Start des Batch-Files (enthält DOS-Kommandos) für JDK-Version 1.1.4

```
k:\start
```

bzw. von JDK-Version 1.2.1

```
k:\start_121
```

- 4) Starte in der DOS-Box/dem Terminalfenster den Java-Compiler "javac":

```
javac Bsp_02.java
```

Es entsteht das File "Bsp_02.class", das den portablen Bytecode enthält.

- 5) Erstelle mit einem Texteditor ein Textfile beliebigen Namens, z.B. "Test.htm" oder "Test.html" mit HTML-Kommandos und der Dateiendung ".htm" bzw. ".html" im selben Verzeichnis "C:\work" (Linux-User: "\${HOME}/work"). Das HTML-File muß im BODY-Teil den APPLET-Befehl beinhalten:

```
<APPLET CODE="Bsp_02.class" HEIGHT=300 WIDTH=500>  
  Start des Java-Applets gescheitert! Browser-Einstellungen kontrollieren  
  (Start von Java-Programmen erlaubt?) oder einen Java-f&auml;higen Browser  
  besorgen!  
</APPLET>
```

Als Wert für CODE muß dabei der Name des Files "Bsp_02.class" mit dem Java-Bytecode enthalten. WIDTH und HEIGHT spezifizieren Breite und Höhe des Applets in der HTML-Darstellung des Browsers. Der Text innerhalb des APPLET-Befehls wird nur sichtbar, wenn der Start des Java-Applets scheitert. Appletfähige Browser interpretieren nur noch den PARAM-Befehl innerhalb des APPLET-Befehls.

- 6a) Starte Netscape oder den Internet Explorer und öffne das lokale HTML-File "Test.htm" bzw. "Test.html" im Browser. Beim Darstellen des HTML-Files stößt der Browser auf den APPLET-Befehl und startet daraufhin das Java-Applet und setzt danach die Interpretation des HTML-Files fort.

- 6b) Starte in der DOS-Box/dem Terminalfenster (aktuelles Verzeichnis wie in 3.) den Appletviewer "appletviewer":

```
appletviewer Test.html
```

Dabei werden jedoch alle HTML-Kommandos außer dem APPLET-Befehl ignoriert. Das Applet startet jedoch im Appletviewer.

import-Anweisung

Das Java-Programm für das Applet kann noch folgendermaßen modifiziert werden. Um anzudeuten, daß der kürzere Name "Applet" für die Klasse "java.applet.Applet" steht, d.h. für die Klasse "Applet" im Paket "java.applet", kann man eine "import"-Anweisung verwenden. Vor allem, wenn Klassennamen öfter angegeben werden müssen, ist eine kürzere Schreibweise erwünscht.

```
import java.applet.Applet;
import java.awt.*;

public class Bsp_02 extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Java startet als Applet!", 20, 50);
    }
}
```

Die "import"-Anweisung "import java.awt.*" führt für alle Klassen im Paket "java.awt" diese abkürzende Schreibweise ein, insbesondere für die Klasse "Graphics".

mögliche Fehlerquellen beim Schreiben/Starten von Applets:

- Die öffentliche Klasse heißt nicht genauso wie das File (ohne die Endung ".java").
- Der Name des Java-Bytecodes (Endung ".java") im HTML-File ist falsch.
- Bei der Klassendefinition wurde "public" vergessen.
- Bei der Klassendefinition wurde "extends java.applet.Applet" bzw. die Kurzform "extends Applet" vergessen.
- Die Kurzform "extends Applet" wurde verwendet, die zugehörige "import"-Anweisung aber vergessen (Vergleichbares gilt für "Graphics").
- Die erste Zeile der "paint()"-Funktion wurde nicht genauso wie im Beispiel spezifiziert.
- Der Browser ist nicht Java-fähig bzw. der Start von Java-Programmen wurde in der Einstellung des Browsers untersagt (Problem tritt beim Appletviewer nicht auf).

- Der Browser kommt mit der im Java-Applet verwendeten JDK-Version nicht zurecht (Browser verwendet veraltete Java-Version; Problem tritt beim Appletviewer nicht auf).
In Netscape kann man zur besseren Fehlersuche die Java-Console öffnen, im Internet Explorer muß man bei älteren Versionen das Anlegen eines Logfiles beim Start von Java-Programmen veranlassen und dieses auswerten oder bei Version 4.x die Java-Konsole aktivieren. Typische Fehlermeldungen beinhalten dann meistens die Worte "class/interface/method not found".
- Das Java-Applet verletzt die Sicherheitsbestimmungen von Java (versucht z.B. ein lokales File zu lesen oder zu schreiben) und scheitert deshalb, d.h. der Start des Applets wird abgebrochen (siehe Java-Console oder Java-Logfile). Typische Fehlermeldungen beinhalten dann meistens die Worte "security exception".
- Beim Start des Appletviewers wurde versehentlich das Java-Bytefile "Bsp_02.class" angegeben, d.h. er wurde mit dem Befehl

`appletviewer Bsp_02.class`

gestartet.

- Das Java-Programm ist eine Java-Applikation und gar kein Java-Applet.

Erläuterungen (Details):

<code>extends java.applet.Applet</code>	Die Klasse "Bsp_02" erbt die Datenelemente und die 22 Methoden der Java-Klasse "java.applet.Applet", d.h. ein Objekt der Klasse "Bsp_02" kann auch auf die geerbten Datenelemente und Methoden zugreifen. Es kann aber auch diese geerbten Methoden überschreiben durch eigene Varianten (hier z.B. die "paint()"-Methode).
<code>java.awt.Graphics</code>	<code>g</code> ist ein Objekt der Java-Klasse "java.awt.Graphics" und stellt den sog. Grafikkontext dar, der der Zeichenfläche der grafischen Komponente (Container, Canvas, ...) entspricht. Eine Klassenmethode ist dabei "void drawString(String s, int x, int y)".
<code>drawString("Hello, world!", 20, 50);</code>	zeichnet den String "Java startet als Applet!" auf den Grafikkontext des Applets an der Position (x,y) = (20, 50). Dabei ist der Ursprung des Koordinatensystems (0,0) die linke obere Ecke, positive x-Werte sind rechts vom Ursprung, positive y-Werte unterhalb des Ursprungs.

Der Umgang mit Java im Browser