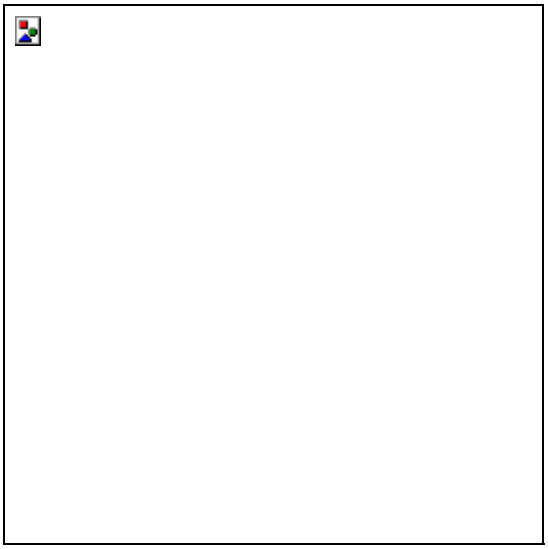
Aufbau und Anwendungspotentiale von Java

Tutorial zum systemtechnischen Workshop "Internet und WWW" bei Prof. Dr. J. Scherff im Fachbereich Wirtschaftsinformatik an der Fachhochschule Furtwangen vorgelegt von:

Mladen Kadic
Andreas Mandel
Heiko Munz

SS 1996

Dieses Dokument wurde mehr oder weniger (leider eher weniger) automatisch aus dem Referatsdokument generiert. Daher sind einige Tabellen und Hervorhebungen etwas mißraten. Leider lies sich das, mit vertretbarem Aufwand nicht mehr anpassen. Wir bitten daher etwas um Nachsicht. Innerhalb des Dokuments gibt es keinerlei links nach außen. Beispielapplets und Links gibt es an anderer Stelle. Zu diesem Tutorat wurde ein paar kleine Übungen entworfen.

Aufgrund unserer wunderschoenen Seiten haben wir die Auszeichnung "Heisse Java Seite!" von "Kaffee&Kuchen" erhalten! 

Inhalt

---

5 Aufbau und Anwendungspotentiale von Java

5.1 Aufbau und Konzepte

In diesem Kapitel soll ein Überblick über die Konzepte von Java gegeben werden. In der Knappheit, die diesem Dokument auferlegt wurde, können natürlich nicht alle Aspekte mit der nötigen Tiefe erörtert werden. Es wurde jedoch versucht alle Besonderheiten der Sprachkonzepte Javas zu würdigen.

5.1.1 Kurzeinführung in JAVA

Was ist JAVA?

JAVA ist eine Programmiersprache.

Beim Design von Java wurde versucht die Goodies der bekannten Programmiersprachen zu übernehmen und deren Nachteile die sich im Laufe deren Existenz gezeigt haben auszubügeln. Java ist rein objektorientiert und hat keine strukturellen Überbleibsel, wie das z.B. in C++ der Fall ist. Java sollte alles enthalten, was das Programmieren angenehm macht und dabei mit einer klar überschaubaren und redundanzfreien Syntax auskommen.

Nach SUN (*http://java.sun.com/java.sun.com/allabout.html*):

Java is a simple, robust, object-oriented, platform-independent multi-threaded, dynamic general-purpose programming environment. It's best for creating applets and applications for the Internet, intranets and any other complex, distributed network.

Java entstand aus der deprimierenden Erfahrung Suns, daß die vorhandenen “neuen” Programmiersprachen mit neuen Programmierparadigmen keinesfalls eine revolutionäre Erleichterung in der Softwareentwicklung mit sich brachten. Java soll nun endlich die Vorzüge neuer Entwicklungsmethoden auf die Ebene der Programmiersprache

bringen und dem Programmierer das “Leben” erleichtern.

Was kann JAVA?

Java kann nicht mehr und nicht weniger als andere Programmiersprachen auch. Durch die Konzeption eignet sich Java jedoch für eine Fülle von Anwendungen die Plattformunabhängigkeit macht Java z.B. zur idealen Programmiersprache für verteilte Anwendungen, insbesondere im WWW. Dabei bleibt der Anspruch auf Portabilität nicht im leeren Raum (man erinnert sich noch vage an die Aussage das C eine portable Programmiersprache ist) , der praktische Einsatz verschiedener Java Applikationen und insbesondere der Applets zeigt, daß Java durch seine Konzepte wirklich auf den unterschiedlichsten Plattformen lauffähig ist.

Verschiedene große Betriebsystem Hersteller wollen Java als Bestandteil in ihre Betriebsysteme integrieren. Dazu gehören z.B. Micro$oft, IBM und Apple.

Was kann JAVA nicht?

Kaffee kochen.

5.1.2 Gegenüberstellung Java – herkömmliche Programmiersprachen

Um JAVA gegenüber den bekannten Programmiersprachen abgrenzen zu können werden wir zunächst Unterschiede und Gemeinsamkeiten heraus arbeiten und dann auf die Unterschiede genauer eingehen.

	Cobol	**C**	**C++**	**Smalltalk**	**Java**
Objektorientiert					
Datentypen					
Zeiger					
Unions/Strukturen					
Einfach					
Standard Library					
Portable					
Garbage Collection					
Threading					
Fehlerbehandlung					
Precompiler					
Interpretiert					
Linker					
Dynamisch					
Verifizierung					
Robust					
Sicher					

Tabelle : Java und andere Programmiersprachen (left to the reader)

5.1.3 Was fällt weg? (im Vergleich zu C/C++)

Keine Strukturen, Aufzählungstypen, Unions

Die Aufzählungstypen (z.B. *enum*), Strukturen und Unions können in Java, wie in jeder objektorientierten Programmiersprache, durch geeignete Klassen dargestellt werden. Deshalb sind explizite Typendefinitionen unnötig. Dieser Schritt, *typedefs*, *enums* etc. aus C++ zu entfernen war wegen der gewünschten "Kompatibilität" zu C nicht möglich. In Java wird durch diese Maßnahme die Syntax wesentlich schlanker und auch die Probleme der Namensraumvergabe wird durch die dann nötigen zugehörigen Klassen vermieden. (In C haben z.B. alle *enum* Typen einen eigenen, gemeinsamen Namensraum)

Keine Funktionen

Alleinstehende Funktionen, die nicht Methoden einer Klasse sind, werden nicht unterstützt. Das vermeidet Mehrdeutigkeiten und Widersprüche in der Klassenhierarchie. Java unterstützt Klassenmethoden und Klassenvariablen, wodurch *alleinstehende* Funktionen unnötig werden.

Keine Header Files

In Java werden die Klassen komplett in einem File Codiert. Header Files wie in C gibt es nicht. Zu jeder Klasse wird ein eigenes Klassenname.*class* File erzeugt. Dadurch kann es z.B. nicht mehr vorkommen, daß durch Änderungen in einer Zentralen Klasse in einem Header File das gesamte Projekt neu Übersetzt werden muß. Der Einsatz von Java macht Hilfsmittel wie *make* und trickreiche Compilerfeatures, die versuchen dieses Problem zu minimieren unnötig.

Kein Überladen von Operatoren

Java unterstützt kein Operator Overloading. Diese Möglichkeit war bzw. ist schon immer umstritten. Sun hat der Regel der Einfachheit und Klarheit Vorrang gegeben. Die Möglichkeiten des Operator Overloadings kann durch Klassenmethoden einfach ersetzt werden.

Kein Precompiler (z.B. #DEFINE)

Precompiler, wie man sie von C oder C++ her kennt, werden von Java nicht eingesetzt. Der durchgängig objektorientierte Ansatz soll einen Precompiler unnötig machen. Globale Definitionen von Konstanten können als Klassenvariablen (*static*) realisiert werden. Dadurch bekommen die Konstanten auch einen eigenen Namensraum, was widersprüchliche Bezeichnernamen ausschließt. Diverse Tricks und auch Fehlerquellen werden so ausgeschlossen.

Keine vage definierte Datentypen

In Java sind die primitiven Datentypen, anders als in C vollständig definiert. So ist ein *int* immer 32 Bit breit und mit Vorzeichen behaftet. Es gibt anders als in C **feste** Definitionen für die Basis Datentypen. Die übrigens die einzigen Typen sind die keine Klasse darstellen. Durch Vererbung können ihnen jedoch weitere Methoden zugeordnet werden.

Die direkte Implementierung von Basis Datentypen in die Programmiersprache verbessern das Laufzeitverhalten, gegenüber Typenlosen Programmiersprachen wie zum Beispiel Smalltalk, enorm.

Keine (expliziten) Pointer und keine Pointerarithmetik

Java hat zum Ziel, eine möglichst sichere Laufzeitumgebung zur Verfügung zu stellen. Aufgrund der Möglichkeit, komplexe Strukturen mit Hilfe von Klassenhierarchien darzustellen, ist Java nicht auf Pointerarithmetik angewiesen. In Java gibt es generell keine expliziten Pointer, womit ein weiterer Großteil der Fehlermöglichkeiten herkömmlicher C und C++ Programme ausgeschlossen werden.

Keine vargs (freie Argumentenanzahl)

Java unterstützt keine Funktionen mit variabler Argumentenanzahl wie es z.B. in C bei der Funktion *printf* verwendet wird. Auch hier stellt der objektorientierte Ansatz geeignetere und weniger fehlerträchtige Möglichkeiten zur Verfügung (wie es schon in C++ mit *cout* realisiert wird).

Keine Mehrfachvererbung

Die gewaltigen Probleme die durch den Einsatz von Mehrfachvererbungen auftreten können sind dadurch umgangen

werden, daß es diese Möglichkeit nicht gibt. Durch den Einsatz von Interfaces die einen Satz von Schnittstellenmethoden definieren, die in der Klasse implementiert werden, wird eine weitgehend vergleichbare Funktionalität ohne die bekannten Probleme erreicht. Diese Interfaces wurden den *protocol*s in Objective C nachempfunden.

Kein Semikolon oder doch Semikolon

Beim Design von Java wurde darauf geachtet, daß der Programmierer nicht mit irgendwelchen "; erwartet" Fehlermeldungen zu kämpfen hat. Wo es die Grammatik erlaubt ist der ";" optional oder kann beliebig oft gesetzt werden. Bei der Entwicklung der Sprache wurde auch darauf geachtet, das der Compiler die Möglichkeit hat Fehler genau zu lokalisieren. Die in C übliche Fehlermeldung mit Verweis auf eine Zeile die auf die eigentlich fehlerhafte Zeile mit fehlendem Semikolon folgt sollte in Java nicht vorkommen.

5.1.4 Speicherverwaltung

Als erste Besonderheit, insbesondere für nicht Smalltalk Programmierer ist das Konzept der Speicherverwaltung in Java zu nennen. In Java gibt es keinen Heap auf dem dynamische Daten allociert werden, also auch kein *malloc* oder *mfree.* Vielmehr sorgt die Java Laufzeitumgebung selbst dafür, das nicht mehr benötigte Objekte automatisch freigegeben werden. Dazu trägt jedes Objekt einen Zähler mit sich, wie viele Verweise auf das Objekt noch existieren.

Die Arbeit selbst dafür zu sorgen das Objekte rechtzeitig freigegeben werden, die z.B. in C++ bei komplexeren Programmen extreme Ausmaße annimmt entfällt also völlig. Man spricht von einem Garbage Collector. In Java läuft dieser Garbage Collector in einem eigenen Task mit niederer Priorität, das Aufräumen wird also im Hintergrund erledigt.

Speicherleichen, also Objekte die nicht mehr benötigt werden, deren Speicher jedoch nicht freigegeben wurde gehören automatisch der Vergangenheit an. Es kann auch nicht mehr vorkommen das eine Referenz auf ein nicht mehr allociertes Objekt existiert, was bei C und C++ zu häufigen Fehlern geführt hat (z.B. eine durch eine lokale Funktion deklarierte Variable wird per Pointer dem Aufrufer bekanntgegeben).

Dieses Konzept tritt nicht nur auf Objekte sondern auch für Klassen in Kraft, da Klassen in Java nichts anderes sind als von der Klasse Objekt abgeleitete Objekte. Das heißt insbesondere das zur Laufzeit eines Programms nur die Klassen im Hauptspeicher gehalten werden die auch wirklich gebraucht werden. Das hat zur Folge, daß beim Programmstart zunächst nur die Basisklassen z.B. der awt Library geladen werden, was allerdings auf Grund des dynamischen Linkvorgangs, der gleichzeitig einsetzt merklich Zeit in Anspruch nimmt.

5.1.5 Threads

Java verwendet selbst z.B. zur Realisierung der Speicherverwaltung Threads und es bietet dem Programmierer Klassen und Methoden zur Threadbildung zur Verfügung. Threads ermöglichen das parallele Ausführen einzelner Programmteile. Bei GUI basierenden Systemen ist es z.B. wichtig das verschiedene Aufgaben parallel erledigt werden. So will der Anwender in der Menüstruktur blättern, während in einem Fenster Berechnungen ausgeführt werden und in einem anderen gerade eine Grafik aufgebaut wird.

Bei Java muß sich der Programmierer nicht mehr mit den Besonderheiten der Zielplatform auseinandersetzen und er muß auch keine Bedenken haben ob die standard Library Funktionen reentredant (sprich mehrfach gleichzeitig aufgerufen werden dürfen) sind. Er bekommt durch Java einheitliche Methoden zur Synchronisation von Threads und zur Kommunikation zwischen verschiedenen Threads zur Verfügung gestellt. Durch den Modifizierer *synchronized* können Methoden eines Objekts gegen mehrfachen parallelen Aufruf geschützt werden. Ein in der Java Laufzeitumgebung implementierter Monitor überwacht dann das diese Methoden des Objekts nicht parallel sondern seriell Ausgeführt werden. Generell sollten alle Methoden die Objektatribute verändern als *synchronized* deklariert werden.

Die Standardbibliothek ist vollständig "Threadfest", das heißt Methoden kann uneingeschränkt mehrfach parallel aufgerufen werden.

5.1.6 Benennungskonventionen

Wie schon mehrfach erwähnt werden in Java Klassen unter ihrem Realnamen abgelegt. Dadurch muß man beachten, daß ein File mit dem Namen *test.java* nach der Compilierung unter Umständen mehrere Files erzeugt die zum Beispiel *klasse1.class, klasse2.class* etc. heißen. Zum Auffinden der Klassen wird vom ClassLoader der Klassen

eine bestimmte Strategie verfolgt (wie Eingangs erwähnt kann der ClassLoader auch überladen werden und dadurch sein Verhalten verändert werden). Klassen werden zu sogenannten Packages zusammengefaßt. Bestes Beispiel sind die standard Librarys deren Namen alle mit dem Prefix *java.* beginnen. Um den Anspruch an die Dateisysteme nicht übergebühr zu erhöhen wird zum Beispiel für die Klasse *java.awt.list* der Filenamen *list.class* Verzeichnis .*java**awt*\ verwendet.

Inzwischen ist man dazu übergegangen die kompletten Librarys in einem Archiv zu speichern (*CLASSES.ZIP*). Dieses File darf bei der Installation von Java keinesfalls ausgepackt werden.

Zur Benennung von Klassen die (Inter-) Netz weit bekannt sein sollen, gilt die Konvention als Prefix der Klassen den eigenen URL zu verwenden (dadurch wäre es mit einem geeigneten ClassLoader möglich Klassen automatisch in der aktuellsten verfügbaren Version vom durch den URL zu identifizierenden Server zu Laden). Die Problematik der unter Umständen recht langen Klassennamen (*de.fh-furtwangen.ask.katze.libs.car.class*) wird derzeit heftig diskutiert und ist auch reichlich umstritten.

5.1.7 Datentypen und Strukturen

In Java gibt es folgende grundlegende Basis Datentypen:

- *byte* 8 Bit Zahlenwert (-128..127)
- *short* 16 Bit Zahlenwert
- *int* 32 Bit Zahlenwert
- *long* 64 Bit Zahlenwert
- *float* 32 Bit IEEE 754 Fließkommazahl
- *double* 64 Bit IEEE 754 Fließkommazahl
- *char* 16 Bit Unicode Zeichen
- *boolean* “echter” 1 Bit Wert
- Arrays
- *String* Unicode String

Dabei kommt der Programmierer nicht mit den Endiantypen der numerischen Variablen in Berührung. Was z.B. unter C zu Problemen beim Dateiaustausch führt.

String ist in Java ein echter Datentyp und nicht wie in C ein künstliches Array von Zeichen. Allerdings sind einige Methoden zur Stringmanipulation in einer Klasse der Standard Library gekapselt.

5.1.8 Die Syntax, eine Übersicht

Die Kontrollstrukturen, Schleifen und Operatoren sind weitgehend die von C oder C++ her bekannten Konstrukte. Lediglich bei den Operatoren mußte, da es keine vorzeichenlosen Zahlen gibt der >>> Operator eingefügt werden, der eine Zahl bitweise nach rechts Schiebt und von rechts Nullen nach schiebt. Im Gegensatz zum >> Operator, der das Vorzeichenbit nach schiebt. In C wurde dies anhand dessen entschieden, ob der Zahlenwert vorzeichenbehaftet war oder nicht.

Auf der nächsten Seite befindet sich die, bis auf die *kursiv* dargestellten Symbole DocComment, Identifier, Number, String und Character komplette Syntaxbeschreibung der Programmiersprache Java. Die Syntax ist trotz der Objektorientiertheit nicht wesentlich Komplexer als die der Programmiersprache C. Die **Fett** dargestellten Symbole sind Zeichen wie sie im Quelltext erscheinen. Folgende Notation wird für die Grammatik verwendet:

Ausdruck = Metaausdruck; “|” steht für Alternativen, (...) für Gruppierungen, ? für kein oder einmaliges Auftreten und * für kein oder mehrmaliges Auftreten.

```
CompilationUnit= PackageStatement? ImportStatement* TypeDeclaration* ;
```

PackageStatement= **package** PackageName **;** ;

ImportStatement= **import** PackageName **. * ;**| **import** (ClassName | InterfaceName); ;

TypeDeclaration= ClassDeclaration | InterfaceDeclaration | **;** ;

ClassDeclaration = Modifier* **class** *Identifier* (**extends** ClassName)? (**implements** InterfaceName (**,** InterfaceName)*)? **{** FieldDeclaration* **}** ;

InterfaceDeclaration = Modifier* **interface** *Identifier* (**extends** InterfaceName (**,** InterfaceName)*)? **{** FieldDeclaration* **}** ;

FieldDeclaration = *DocComment*? MethodDeclaration | DocComment? ConstructorDeclaration | *DocComment*? VariableDeclaration | StaticInitializer | **;** ;

MethodDeclaration = Modifier* Type *Identifier* **(** ParameterList? **)** (**[ ]**)* (**{** Statement* **}** | **;**) ;

ConstructorDeclaration = Modifier* *Identifier* **(** ParameterList? **) {** Statement* **}** ;

VariableDeclaration= Modifier* Type VariableDeclarator (**,** VariableDeclarator)* **;** ;

VariableDeclarator= *Identifier* (**[ ]**)* (**=** VariableInitializer)? ;

VariableInitializer= Expression | **{** (VariableInitializer (**,** VariableInitializer)* **,**?)? **}** ;

StaticInitializer= **static {** Statement* **}** ;

ParameterList = Parameter (**,** Parameter)* ;

Parameter = TypeSpecifier *Identifier* (**[ ]**)* ;

Statement = VariableDeclaration | Expression **;** | **{** Statement* **}** | **if (** Expression **)** Statement (**else** Statement)? | **while (** Expression **)** Statement | **do** Statement **while (** Expression **) ;** | **try** Statement (**catch (** Parameter **)** Statement)* (**finally** Statement)? | **synchronized (** Expression **)** Statement | **return** Expression? **;** | **throw** Expression **;** | **switch (** Expression **) {** Statement* **}** | **case** Expression **:** | **default**; | *Identifier* **:** Statement | **break** *Identifier*? **;** | **continue** *Identifier*? **;** | **;** ;

BiOp= **+** | **-** | ***** | **/** | **%** | **^** | **&** | **|** | **&&** | **||** | **<<** | **>>** | **>>>** | **=** | **+=** | -= | ***=** | **/=** | **%=** | **^=** | **&=** | **|=** | **<<=** | **>>=** | **>>>=** | **<** | **>** | **<=** | **>=** | **==** | **!=** | **.** | **,** ;

Expression= Expression BiOp Expression | Expression **instanceof** (ClassName | InterfaceName) | Expression **?** Expression **:** Expression | Expression **[** Expression **]** | **++** Expression | **--** Expression | Expression **++** | Expression **--** | **-** Expression | **!** Expression | **~** Expression | **(** Expression **)** | **(** Type **)** Expression | Expression **(** ArgList? **)** | **new** ClassName **(** ArgList?**)** | **new** TypeSpecifier (**[** Expression **]**)+ (**[ ]**)* | **new (** Expression **)** | **true** | **false** | **null** | **super** | **this** | *Identifier* | *Number* | *String* | *Character* ;

ArgList = Expression (**,** Expression)* ;

Type = TypeSpecifier (**[ ]**)* ;

TypeSpecifier= **boolean** | **byte** | **char** | **short** | **int** | **float** | **long** | **double** | ClassName | InterfaceName ;

Modifier= **public** | **private** | **protected** | **static** | **final** | **native** |

```
synchronized | abstract | threadsafe | transient ;

PackageName= Identifier | PackageName . Identifier ;

ClassName= Identifier | PackageName . Identifier ;

InterfaceName= Identifier | PackageName . Identifier ;
```

Java läßt für Bezeichner (*Identifier*) Zeichen aus dem Unicode zu, das heißt, daß z.B. auch Umlaute in Namen vorkommen können. Diese Möglichkeit bereitet allerdings noch etlichen Dateisystemen Probleme, da die Klassennamen direkt in **.class* Files abgelegt werden, daher ist bei der Benennung der Klassen darauf zu achten nur ASCII Zeichen zu verwenden, die auch in üblichen Dateisystemen verwendet werden dürfen. Da einige Betriebsysteme etwas lausig mit dem Vergleich von Dateinamen umgehen (Namentlich z.B. WIN95) sollte auch darauf geachtete werden, daß sich Klassen nicht nur durch Groß- und Kleinschreibung unterscheiden, was in Java generell möglich ist.

Zahlen (Number) werden wie in C üblich dargestellt. char ist immer ein 16Bit Unicode Wert, der entsprechend dargestellt wird 7 Bit ASCII Werte werden direkt in ihre Unicode Entsprechungen transformiert.

Strings werden wie in C üblich durch doppelte Hochkomma eingeschlossen dargestellt. Für die Darstellung von Unicode Zeichen gibt es eine spezielle Escape Darstellung.

Bleiben noch die DocComments, mittels des zur Java Entwicklungsumgebung gehörenden Dokumentencompilers kann aus Java Quelltext eine Dokumentation der Klassenhierarchie und der Klasseninterfaces erstellt werden. Sun hat aus der Not, das der Quelltext sich schneller verändert als die Dokumentation angepaßt wird eine Tugend gemacht und ermöglicht die Dokumentation des Codes direkt im Programmcode unterzubringen.

Um textuelle Beschreibungen in diese Dokumentation einfließen zu lassen gibt es eine spezielle Kommentarklammerung. Der Text zwischen /** und */ wird zur Beschreibung des darauffolgenden Codes (in der Regel eine Definition) verwendet und Automatisch eingebunden. Das erzeugte Dokument wird im HTML Format ausgegeben. Es ist auch möglich im Kommentar HTML Tags zu verwenden die dann in das Dokument übernommen werden. Hyperlinks ermöglichen es die Verschiedenen Hierarchiestufen der Klassen zu verfolgen. In den Kommentaren können dadurch auch HTML Tags z.B. zur Hervorhebung genutzt werden.

5.1.9 Compiler

Der Java Compiler (*javac*) überprüft den Quelltext und erzeugt aus den **.java* Files die entsprechenden **.class* Files der Klassen die darin Implementiert sind. In der aktuellen JDK Version (1.01) muß zwischen dem Dateinamen des Quelltextes und den der erzeugten Class Dateien kein Zusammenhang mehr bestehen. In früheren Versionen mußte in der Quelltextdatei eine Klasse mit dem Namen der Datei enthalten sein. Bei der Compilation wird der Quelltext bereits, soweit möglich, auf Fehler untersucht und in den von der Virtuellen Maschine zu interpretierenden Bytecode umgewandelt. Identifizierer, die zum Beispiel für den Zugriff auf andere Klassen benötigt werden bleiben dabei in Klarschrift erhalten und werden erst vom Interpreter in (physische) Adressen umgesetzt.

Zum Compiler gehört auch *javadoc*, der aus Java Quellfiles ein HTML Dokument der Klassenhierarchien und der Klassen Methoden und Attribute erzeugt. Dazu werden auch die Texte aus den Dokumentkommentaren verwertet.

In Suns JDK (Java Development Kit) ist auch ein Compiler namens *javac_g* vorhanden. Dieser hat die gleiche Funktionalität wie der *javac* Compiler, er wurde jedoch mit Debuging Informationen übersetzt und ermöglicht so das leichtere Auffinden von Fehlern *innerhalb* des Compilers. Der Compiler selbst verwendet die Java Librarys, daher ist die Größe des Programms überraschend gering.

5.1.10 Virtuelle Maschine

Die Virtuelle Maschine interpretiert den Java Bytecode. Erst sie löst externe Referenzen auf, indem sie den ClassLoader, der die benötigte Klasse bei Bedarf sucht und in den Hauptspeicher lädt, aufruft und die Klartextnamen durch Adressreferenzen ersetzt.

Auf einem Computer bzw. Betriebssystem auf dem diese Virtuelle Maschine realisiert ist können alle Java Programme laufen. Dabei ist keine Neucompilierung des Quelltextes nötig. Vor der Instanziierung einer Klasse werden dabei Laufzeitprüfungen und eine Verifikation vorgenommen, welche die Sicherheit des Programms gewährleisten sollen (dazu später mehr).

5.1.11 Linker?

In Java gibt es keinen Linker im ursprünglichen Sinne, wie schon oben Beschrieben wird jede Klasse in ein einzelnes File gespeichert. Der ClassLoader lädt die Klassen automatisch bei Bedarf (und erst dann) in den Hauptspeicher. Gibt es keine Referenz bzw. Instanz der Klasse mehr, so wird der Speicher wieder freigegeben.

Der Klassenlader kann dabei durch eigene Klassen überladen werden und ermöglicht so z.B. das auffinden einzelner Klassen über die Grenzen des lokalen Rechners hinweg.

Der Linkprozeß findet also erst zur Laufzeit des Programms statt. Eine Unterscheidung, wie in C++ zwischen frühem und spätem Linken existiert nicht.

5.2 Java-Entwicklung und Java-Tools

5.2.1 Was ist Java?

Java ist eine neue, von Sun Microsystems entwickelte, objektorientierte und plattformunabhängige Programmiersprache. Dabei wird oft davon gesprochen, daß Java das WWW revolutionieren wird. Denn mit Java lassen sich faszinierende Dinge auf Web-Seiten machen, die bisher nicht möglich waren. Herausragenstes Merkmal ist wohl das Ausmaß der Interaktivität, die doch bei weitem alles übersteigt, was bei CGI-programmierten Web-Seiten zur Zeit möglich ist. So eröffnet Java ganz neue Möglichkeiten Web-Seiten mit Multimedia-Inhalten wie flüssige Animationen, erweiterte Grafiken, Sound und Video effektvoll zu gestalten.

5.2.2 Wie entstand Java?

Das Java-Team bei Sun Microsystems wollte ursprünglich Software zur Gerätesteuerung entwickeln. Allerdings mußte das Team feststellen, daß die existierenden Programmiersprachen wie C und C++ dafür nicht geeignet waren.

In C oder C++ geschriebene Programme müssen für einen bestimmten Computerchip kompiliert werden. Kommt ein neuer Chip auf den Markt, so können seine Leistungsmerkmale nur dann voll genutzt werden, wenn der größte Teil der Software erneut kompiliert wird.

Da ein Computerchip, der zu teuer wird, umgehend durch einen neueren, kosteneffizienteren ersetzt wird, muß eben auch die Steuerungssoftware auf dem neuen Chip arbeiten.

So begann 1990 James Gosling bei Sun Microsystems in Mountain View, Kalifornien, mit der Arbeit an einer neuen Programmiersprache, die für Steuerungselektronik geeigneter und gleichzeitig die mit den traditionellen Sprachen wie C und C++ verbundenen Probleme umgehen sollte.

5.2.3 Die ersten Java-Projekte

Die ersten Java-Projekte kamen selbstverständlich von Sun selbst und beeinflußte die Java-Entwicklung nachhaltig.

Das Green-Projekt

Das Green-Projekt war das erste Projekt, bei dem Java eingesetzt wurde. Es sollte eine neue Form von Benutzeroberfläche für die Steuerung von Geräten im häuslichen Alltag wie Videorecorder, Fernseher, Lampen, Telefon usw. entwickelt werden. So bauten die beteiligten Mitarbeiter einen experimentellen, tragbaren Computer, dem sie den Namen “*7” (gesprochen: “star seven”) gaben.

Die Benutzeroberfläche, die vollständig in Java geschrieben worden war, bestand nun aus einer farbigen, animierten Darstellung der häuslichen Umgebung, bei der verschiedene Geräte durch Bildschirmberührung bedient werden konnten.

Das Nachfolgeprojekt für den *7 war ein Demoprogramm für Video-On-Demand (VOD). Hier sollten kleine Programme über die TV-Kanäle übertragen werden, um das Fernsehen interaktiv und attraktiv zu gestalten. Dadurch konnte gezeigt werden, daß sich die Benutzeroberfläche (und natürlich Java) nicht nur zur Gerätesteuerung eignet, sondern auch für interaktives Fernsehen.

WebRunner und HotJava

Etwa 1993 entwickelte sich das World Wide Web von einem auf Text basierenden zu einem immer stärker grafisch ausgerichteten Interface, das immer mehr Interesse fand.

Da Java-Programme durch die Plattformunabhängigkeit auf all den verschiedenen Computertypen wie PC, Mac oder Unix ablauffähig sind und deren lokale Rechenpower nutzen, wurde den Java-Entwicklern klar, daß sich diese Sprache hervorragend für die Programmierung von Web-Anwendungen eignet, ja sogar eine neue Dimension eröffnet.

So entwickelte man einen Web-Browser vollständig in Java und nannte ihn “WebRunner”. Aus urheberrechtlichen Gründen wurde dieser Browser später in “HotJava” umgetauft. So war “HotJava” der erste Web-Browser, der Java-Applets unterstützte. Dieser Web-Browser brachte nun die Programmiersprache Java erst ins Rampenlicht der Öffentlichkeit.

Im Mai 1995 stellte Sun Microsystems die Java-Technologie in San Francisco offiziell vor. Dadurch, daß Netscape Communications zusätzlich bekanntgab, daß der Netscape Navigator Java-Applets unterstützt, hat das schon ohnehin sehr große Interesse nur noch eingeheizt. So unterstützt der Netscape Navigator ab der Version 2.0 Java-Applets.

5.2.4 Das Java-Development-Kit (JDK) und dessen Nutzung

Das JDK wird von Sun Microsystems für verschiedene Plattformen kostenlos angeboten. So können mit dem JDK “Java Applets” als auch “Java Applications” (Stand-alone-Anwendungen, die mit dem Interpreter “java” zu starten sind) entwickelt werden. Aber auch andere Unternehmungen, wie etwa IBM, entwickeln an einem JDK für ihre jeweiligen Systeme. So steht inzwischen z.B. für OS/2 ein JDK 1.0 und ein Java-fähiger WebExplorer jeweils im Betastadium zur Verfügung. Anm.: Den WebExplorer kann man unter “http://www.ics.raleigh.ibm.com” beziehen. Anm.: Daß Java wohl keine Eintagsfliege ist, machen die Ankündigungen mehrerer großer Betriebssystemhersteller deutlich, daß sie in Zukunft Java in ihre Betriebssysteme integrieren wollen.

Wie man das JDK installiert und nutzt soll nun am Beispiel von OS/2 erläutert werden:

Das JDK für OS/2 besteht aus zwei gepackten Archiven:

- Runtime.zip und
- Toolkit.zip

Um die Archive zu entpacken, wechselt man mittels “cd” in ein Verzeichnis, welches das JDK erhalten soll und kopiert obige Dateien hinein.

Wenn man also z.B. das JDK auf Laufwerk C: im Verzeichnis C:\Prg (in welchem mehrere Programmiersprachen installiert sind) installieren möchte, kopiert man eben dort die Archive hinein.

Nun entpackt man beide Archive. Dabei ist zu beachten, daß der Entpacker lange Dateinamen unterstützen muß und daß man den Parameter zur Generierung der Unterverzeichnisse nicht vergessen darf! Der Aufruf sieht dann z.B. folgendermaßen aus:

```
C:\Prg>unzip -d runtime
C:\Prg>unzip -d toolkit
```

Es wurde nun folgendes Unterverzeichnis erstellt:

```
C:\Prg\JavaOS2
```

In diesem Verzeichnis befinden sich nun alle benötigten Programme wie etwa Compiler, Interpreter, Appletviewer usw.

Es wurde folgende Verzeichnisstruktur erstellt:

```
Datenträger, Laufwerk C, hat den Namen OS2.
Datenträgernummer ist 667C:8C15
Verzeichnis von C:\prg\javaos2
2.05.96 14.19 <DIR> 0 .
```

```
2.05.96 14.19 <DIR> 0 ..
12.04.96 10.58 <DIR> 0 bin (Java-Tools)
12.04.96 11.00 <DIR> 0 demo (Beispiele)
12.04.96 10.58 <DIR> 0 dll
12.04.96 11.00 <DIR> 0 include (C/C++-Einb.)
12.04.96 10.58 <DIR> 0 lib (Java-Packages...)
12.04.96 0.15 483 0 copyrght
2.05.96 14.25 1068 0 index.html
12.04.96 0.15 3119 0 readme.run
12.04.96 0.15 5657 0 readme.tlk
12.04.96 10.55 3564 0 release.not
12 Datei(en) 13891 Byte belegt
48381440 Byte frei
```

So belegt das JDK für OS/2 etwas mehr als 7 MB auf dem Plattenspeicher (ohne Beispiele!).

Um nun damit arbeiten zu können, müssen noch einige Änderungen an der Systemdatei “Config.sys” gemacht werden:

- das PATH-Statement muß um C:\Prg\JavaOS2\bin erweitert werden
- das LIBPATH-Statement muß um C:\Prg\JavaOS2\dll erweitert werden
- SET JAVA_HOME=C:\Prg\JavaOS2
- SET JAVA_USER=C:\Daten\Internet\JavaOS2 (Hier werden benutzerspezifische Daten gespeichert. Dieses Verzeichnis sollte von dem Programmverzeichnis verschieden sein!)
- SET JAVA_WEBLOGS=C:\Daten\Internet\JavaOS2 (Hier wird für jeden gestarteten Java-Prozeß ein Logfile erstellt. Dieses Verzeichnis sollte von dem Programmverzeichnis verschieden sein!)

Nach einem Neustart des Rechners stehen nun folgende (grundlegenden) Befehle zur Verfügung:

- **javac** - Das ist der Java-Compiler. Durch Eingabe von “javac DasPrg.java” wird eine Datei “DasPrg.class” erzeugt, welches nun den Java-Bytecode enthält.
- **java** - Das ist der Java-Interpreter für Java-Applikationen, die NICHT das “Abstract Window Toolkit (AWT)” benutzen. Durch Eingabe von “java DasPrg.class” wird die Java-Applikation ausgeführt.
- **javapm** - Das ist der Java-Interpreter für Java-Applikationen, die das “Abstract Window Toolkit (AWT) benutzen. Durch Eingabe von “javapm DasPMPrg.class” wird das Programm auf der Workplace-Shell von OS/2 ausgeführt.
- **applet** - Das ist ein Programm, um Applets in HTML-Sites ohne einen Web-Browser ausführen zu können. Durch Eingabe von “applet WebSite.html” werden evtl. vorhandene Applets innerhalb der vorgegebenen Web-Site auf der Workplace-Shell von OS/2 ausgeführt. Anm.: Auf anderen Plattformen heißt das Programm zum starten von Applets meist “appletviewer”.
- **javadoc** - Dies ist ein Programm, das aus bestimmt markierten Kommentaren automatisch eine Dokumentation im HTML-Format erzeugt. Die Dokumentation des JDK's wurde auf diese Weise erstellt.

5.2.5 Die Java-Packages

Neben Klassen, die es ermöglichen, Felder und Methoden zu gruppieren, existieren in Java die sogenannten Packages. Mit Hilfe dieser Packages ist es möglich, verwandte Klassen zu gruppieren. Klassen aus solchen Packages lassen sich durch einen von Modula-2 abgeleiteten import-Befehl durch andere Klassen benutzen. Der import-Befehle ist damit mit einem #include-Statement bei C oder C++ vergleichbar.

Die Java-Packages

In Java gibt es sechs (sehr umfangreiche) Packages, die standardmäßig zum JDK gehören:

- **java.lang**: Dieses Package beinhaltet wesentliche Java-Klassen. Dieses Package wird implizit in jede Java-

Datei importiert. So muß ein “import java.lang.*” nicht in einer Java-Datei deklariert werden.

- **java.io**: Dieses Package enthält Klassen, die für die Datenein- bzw. ausgabe benutzt werden können.
- **java.util**: Enthält Utility-Klassen wie Hashtables, Vektoren, Zufallszahlen, Datum usw.
- **java.net**: Enthält Klassen für die Netzwerkverbindung. Diese Klassen können zusammen mit den Klassen aus dem java.io-Package benutzt werden, um Daten aus dem Netzwerk zu lesen oder in das Netzwerk zu schreiben.
- **java.awt**: Mit diesem Package lassen sich plattformunabhängige GUI-Applikationen schreiben. Es sei hier ausdrücklich darauf hingewiesen, daß speziell dieses Package Java für die Programmierung im Internet interessant macht. Mit diesem Package erreicht man einen Interaktivitätsgrad, der mit herkömmlicher CGI-Programmierung undenkbar wäre.
- **java.applet**: Enthält Klassen für die Java-Applet-Programmierung. Diese Applets können dann in jedem Java-kompatiblen Web-Browser ablaufen.

Das Package “java.awt”

Das Abstract Window Toolkit (AWT) ist ein GUI-Toolkit, das plattformübergreifend arbeitet. Es beinhaltet natürlich nicht alle Features, die die unterschiedlichen Betriebssysteme bieten, sondern hat eine allgemeine Reihe von Merkmalen, die von den meisten Plattformen unterstützt werden. Es ist eben so, daß man sich bei plattformunabhängiger Programmierung nun mal auf den kleinsten gemeinsamen Nenner einigen muß. Dabei werden die Methoden so weit als möglich auf die darunterliegenden plattformspezifischen GUI-APIs abgebildet.

Da das AWT wohl das wichtigste Package für zukünftige Java-Applikationen und besonders Java-Applets darstellt, sollen dessen Aufbau und Möglichkeiten kurz dargestellt werden.

Zusätzlich sei gesagt, daß das AWT natürlich erweiterbar ist, wobei man dann auch sicher sein kann, daß diese Erweiterungen ebenfalls plattformübergreifend arbeiten.

Die folgende Abbildung zeigt nun die wichtigsten AWT-Klassen in ihrer Hierarchie:

Abbildung : Die wichtigsten AWT-Klassen

- **Component**: Das ist die Basisklasse vieler AWT-Klassen. Der Hauptzweck dieser Klasse besteht darin, etwas mitsamt seiner Lage und Größe auf dem Bildschirm darzustellen.
- **Container**: Das ist die Basisklasse aller Komponenten. Sie selbst kann andere Klassen enthalten. Zu dieser Klasse gehört ein Hilfsobjekt: ein Layout-Manager, der die Komponenten im Container nach bestimmten Grundsätzen ordnet.
- **Window**: Ein Hauptfenster ohne Rahmen.
- Frame: Ein Hauptfenster mit Rahmen. Diese Klasse ist meist mit einem “MenuBar”-Objekt verbunden.
- **Dialog**: Ein Hauptfenster, zum Erstellen von Dialogen. Dieses Fenster kann modal oder nicht modal sein.
- **FileDialog**: Dialog zur Dateiauswahl.
- **Panel**: Dies ist eine von der Container-Klasse abgeleitete Klasse, die in anderen Containern benutzt werden kann. Dadurch können komplizierte Layouts erstellt werden.
- **Button**: Ein GUI-Aktionsknopf wie etwa “Abbruch”.
- **Canvas**: Dies ist eine allgemeine Komponente, die es erlaubt, Eingabebefehle des Benutzers zu zeichnen oder abzufangen.
- **Checkbox**: Ein Feld mit Boolean-Charakter. Dieses Feld kann man setzen oder löschen.
- **Label**: Ausgabe einer Zeichenkette an einer bestimmten Stelle.

- **List**: Scollbare Liste von Zeichenketten.
- **Scrollbar**: Rollbalken, mit denen scrollbare Canvases erstellt werden können.
- **TextArea**: Einfache Texteditierung.
- **Textfield**: Textkomponente mit nur einer Zeile zur Erstellung von Formularen.

Dies ist natürlich nur ein kleiner Anriß der Möglichkeiten, die das AWT bietet.

5.2.6 Entwicklungsumgebungen für Java

Mit dem vorliegenden JDK ist natürlich ein angenehmes Arbeiten aufgrund der Kommandozeilen-Befehle nicht machbar. Dies haben die Software-Hersteller für Entwicklungsumgebungen natürlich erkannt und trimmen nun ihre bereits bestehenden C/C++-Entwicklungsumgebungen auf Java.

Daher sollen hier die derzeit wichtigsten Java-Entwicklungsumgebungen kurz vorgestellt werden:

- Sun Microsystems stellte auf der Cebit `96 eine erste Version ihrer Java-Entwicklungsumgebung “Java Workshop” vor, die selbst in Java geschrieben wurde.
- Rogue Wave Software bietet schon eine komplette, visuelle Java-Entwicklungsumgebung für Win32 und Solaris an. “JFactory” beteht aus einem Applikationsdesigner, einem Codegenerator und dem Compiler. Anm.: “Jfactory” ist unter “http://www.roguewave.com” zu beziehen.
- Symantec bietet schon jetzt “Espresso” für Win32 und “Caffeine” für den Macintosh an. Dabei handelt es sich aber lediglich um eine Einbindung von Sun's JDK in die Oberflächen der jeweiligen Symantec-Compiler. Die integrierte Lösung mit eigenem Compiler, Debugger usw. soll es bald als “Café” folgen. Anm.: Kontakt über “http://www.symantec.com”.
- PowerSoft will mit “Optima++” auf den Markt. “Optima++” setzt zwar weiterhin als primäre Programmiersprache C++ ein, soll aber lernen, Java-Applets zu erzeugen.
- Innovative Software bietet ihr auf Java getrimmtes OEW als ein Reverse/Forward-Engineering- und Designwerkzeug an. Die Version 0.9 von “OEW/Java” ist zudem kostenlos vom Server “www.isg.de” zu beziehen.
- Metrowerks bietet für den Mac den “CodeWarrior 8” an. Der “Code Warrior 9” soll nach Auskunft von Metrowerks für den Mac Mitte Mai erscheinen. Dabei sollen sowohl der Compiler, als auch der Debugger Eigenbauten sein. Anm.: Kontakt über “http://www.metrowerks.com”.
- Natural Intelligence bietet für den Mac die integrierte Java-Entwicklungsumgebung “Roaster” an.
- melonSoft Berlin bietet mit “ACupOfCoffee” den ersten kompletten GUI-Builder für das Java-Abstract-Window-Toolkit (AWT) an. “ACupOfCoffee” ist eine Palette für den NextStep InterfaceBuilder. Anm.: Kontakt über “mailto: suckow@contrib.de”.
- Borland will in naher Zukunft ebenfalls eine integrierte Java-Entwicklungsumgebung anbieten. Diese soll ebenfalls plattformübergreifend sein (durch Entwicklung mit Java) und hofiert unter dem Codename “Latte”.
- Microsoft will Java-Entwicklungstools in die nächste Version von Visual C++ integrieren.

5.3 Applets: Animation und Interaktion im WWW

5.3.1 Java und das WWW

Java animiert WEB Seiten und ermöglicht interaktive und spezialisierte Anwendungen. Hypertext als Basis für die Informationsorganisation bietet ein Vehikel für die Auswahl der Information. Das Common Gateway Interface (CGI) stellt eine Schnittstelle zwischen Web-Seiten im Client-Browser und dem WWW-Server dar. Java hingegen

gibt erstmals die Möglichkeit Animation sowie komplexe Benutzerinteraktion in Echtzeit auf WWW-Seiten darzustellen. Durch Java wird die klassische W3-Hypertext-Seite nicht nur dynamischer als mit Hilfe des CGI; Benutzer, die im WWW auf Java-Programme stoßen, können an vielfältiger Interaktion und Animation teilnehmen, die ausschließlich durch die Phantasie des Java-Entwicklers begrenzt wird.

Ausführbare Elemente in W3-Seiten

Lokale Echtzeitinteraktion und -animation im Client-Browser wird durch ausführbare Inhalte ermöglicht, mit denen der WWW-Designer seine Seiten anreichert. Diese ausführbaren Elemente werden mit HTML-Tags in die Web-Seiten eingewoben. Allerdings kann nicht jeder beliebige Browser diese ausführbaren Inhalte anzeigen. Der Browser Hot Java von Sun Microsystems war der erste Browser, der die Java-Features nutzen konnte. Heute ist auch in Netscape 2.0 die virtuelle Java-Engine integriert. Dabei spielt es keine Rolle, auf welcher Plattform der Java-kompatible Browser läuft.

5.3.2 Java-Applets in WWW-Seiten

Die ausführbaren Elemente in WWW-Seiten werden Applets genannt und mit dem <applet>-Tag in die Hypertext Markup Language eingebunden. Ist der Client-Browser Java-kompatibel wird das Applet auf dem Client-Browser problemlos dargestellt.

Applets können auch ohne Kenntnisse der Programmiersprache Java in eigene Dokumente eingebunden werden. Applets verfügen über eine wohldefinierte Schnittstelle, die über den <applet>-Tag in HTML angesprochen werden kann. Der <applet>-Tag hat den in beschriebenen Grundaufbau. Um ein Applet in eigene HTML-Seiten einbinden zu können, müssen also

- Standort (HTML-Adresse und Verzeichnis z.B. http://www.xyz.de/java/class)
- Name (Name des Applets in ByteCode, also mit Suffix “class”)
- Parametereigenschaften

des gewünschten Applets bekannt sein.

1. `<applet codebase =`**`URL`** `code =` **`Applet-Name.class`** `width =` **`X`** `height =` **`Y`**`>`
2. `<!` *`Optionale Parameter`*
3. `alt = “Alternativer Text für Textbrowser”`
4. `name= “Symbolischer Name zur Identifikation der Applets”`
5. `align =“Ausrichtung: left, top, right, middle, absmiddle, baseline, bottom, absbottom”`
6. `vspace=“Vertikaler Abstand um Applet herum (nur wenn align=left oder right)”`
7. `hspace=“Horizontaler Abstand um Applet, sonst wie vspace”`
8. *`Ab hier appletspezifische Parameter`*
9. `>`
10. `<param name =` **`Parametername1`** `value =` **`Parameterwert1`**`>`
11. `<param name =` **`Parametername2`** `value =` **`Parameterwert2`**`>`
12. `<param name =` **`Parametername3`** `value =` **`Parameterwert3`**`>`
13. `.`
14. `.`

15. `.`
16. `<param name = **ParameternameN** value = **ParameterwertN**>`
17. `Sie benötigen einen <b>Java</b>kompatiblen Browser!`
18. `</applet>`

Abbildung : Grundaufbau des <applet>-Tags. Notwendige Attribute sind unterstrichen.

Auf den SRC-Parameter kann vollständig verzichtet werden, wenn sich das Applet im aktuellen Verzeichnis befindet. Dieser Parameter ermöglicht auch das benutzen von Applets, die sich auf irgendeinem WWW-Server befinden (siehe). In diesem Fall muß dort die jeweilige HTTP-Adresse inclusive Pfad eingegeben werden. Der code-Parameter bestimmt den Namen des Applets. Zeile 9 enthält einen alternativen Inhalt, der nur von Browsern angezeigt wird, die Java-Applets nicht unterstützen.

Abbildung : Applets in HTML-Seiten

WWW-Seiten mit Java-Applets entstehen in folgenden Schritten:

- Entwickler erstellen Java-Sourcecode auf ihren Rechnern
- Der Quellcode wird mit dem Java-Compiler kompiliert
- Als Ergebnis erhält man das plattformunabhänige Applet in ByteCode
- Erstellen einer HTML-Seite mit <applet> Tag und einem Mindestmaß an Parametern
- Ausführen der Seite mit Java-kompatiblem Browser oder mit dem im JDK enthaltenen appletviewer

5.3.3 Operationen beim Abruf einer WWW-Seite mit Applets

Im folgenden wird der Ablauf von der Anforderung bis zur Darstellung einer HTML-Seite mit Applet beschrieben.

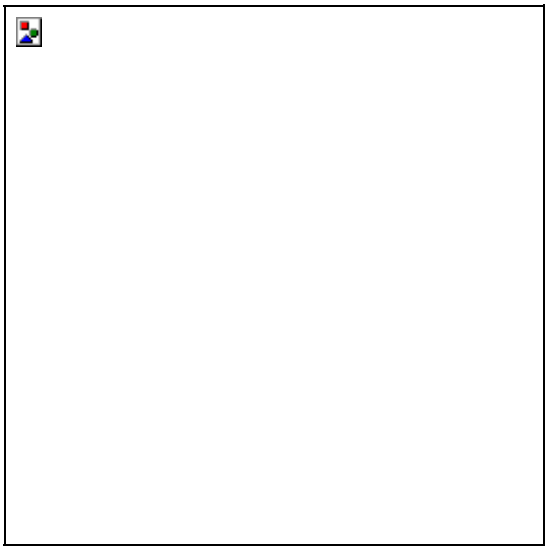


Abbildung : Ablauf von Anforderung bis Darstellung von HTML-Seiten mit Applet

1. Der Benutzer sendet eine Anfrage für ein HTML-Dokument an den Server
2. Das HTML-Dokument wird an den Browser des Benutzers geschickt. Durch die <applet>-Tags wird ein Applet identifiziert.
3. Der Bytecode des Applets wird an den Client übertragen.
4. Der Java-kompatible Client-Browser interpretiert die Bytecodes und zeigt die HTML-Seite mit Applets an

5. Der Benutzer kann unabhängig vom Web-Server mit dem Applet interagieren. Der Bytecode enthält alle dafür notwendigen Informationen. Dabei werden nur Ressourcen des Client-Rechners beansprucht, es sei denn, eine Interaktion mit dem Web-Server würde stattfinden.

5.3.4 Gartner-Group Modell

	Teilhaber-Betrieb	***Distributed Presentation***	***Remote Presentation***	***Distributed Processing***	***Remote Database***	***Distributed Database***	***File Server***
	SERVER	**SERVER**	**SERVER**	**SERVER**	**SERVER**	**SERVER**	**SERVER**
	Data Mgmt.	Data Mgmt.	Data Mgmt.	Data Mgmt.	Data Mgmt.	Data Mgmt.	
	Fachlogik	Fachlogik	Fachlogik	Fachlogik			
	Dialog & Presentation	Dialog & Presentation					
	Internet	Internet	Internet	Internet	Internet	Internet	Internet
						Data Mgmt.	Data Mgmt.
				Fachlogik	Fachlogik	Fachlogik	Fachlogik
		Presentation (phys.)	Dialog & Presentation	Dialog & Presentation	Dialog & Presentation	Dialog & Presentation	Dialog & Presentation
	CLIENT	**CLIENT**	**CLIENT**	**CLIENT**	**CLIENT**	**CLIENT**	**CLIENT**
Java Applets			X	X	X	X	X
CGI			X				
RPC			X				
X11		X					

Abbildung : Applets und CGI (sowie RPC und X11) im Modell der Gartner Group

Java-Applets haben im Vergleich zu CGI-Skripts folgende Vorteile:

- nur notwendige Fachlogik auf Server -> geringer Serverbelastung
- Daten werden da gelagert, wo sie wirklich gebraucht werden
- Nutzung von Client-Ressourcen
- weniger Netzwerkbelastung

5.3.5 Animation und Interaktion mit Java-Applets

Java Applets lassen sich grob nach dem Informationsfluß strukturieren. Untersucht man die Informationsflüße diverser Applets, lassen sich drei Typen bilden.

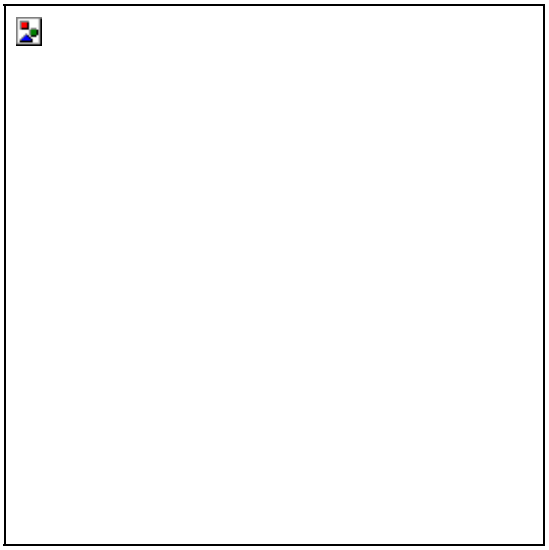


Abbildung : Lokale Präsentation ohne Interaktion

beschreibt den einfachsten Applettyp. Dabei findet ein Informationsfluß vom Server zum Client statt, sowie vom Client zum Benutzer. Ein Beispiel für diese Art von Applet ist das Trembling-Duke Applet sowie alle anderen "Daumenkino"-Applets. Der Bytecode wird lokal ausgeführt, der Benutzer interagiert aber nicht mit dem Applet und es findet kein Infomationsfluß zwischen Client-Browser und Server statt.

Abbildung : Lokale Präsentation und Benutzer-Client Interaktion

Bei dem in dargestellten Applettyp findet nicht nur ein Informationsfluß vom Server zum Client und zum Benutzer statt; der Benutzer hat hier die Möglichkeit mit dem Applet auf dem Client zu Interagieren. Beispielhaft für diesen Typ von Applet ist irgendein Spreadsheet-Applet mit integrierter Fachlogik: Irgendeine Bank bietet ein Spreadsheet in Form eines Applets zur Ratentilgung an. Der Benutzer kann seine persönlichen Daten eingeben und diverse Berechnungen durchführen. Es findet also eine Interaktion zwischen dem Applet und dem Benutzer statt, nicht aber zwischen den Client Browser und dem WWW-Server.

Abbildung : Lokale Präsentation und Server-Client-Benutzer-Interaktion

zeigt einen Applettyp, bei dem Interaktion zwischen allen beteiligten Parteien stattfindet. Beispiel: Eine Versicherung erstellt ein Applet mit dem eine KFZ-Versicherung abgeschlossen werden kann. Das Applet wird bei Anforderung der WWW-Seite auf den Client geladen. Der Benutzer interagiert mit dem Applet indem er seine persönlichen Daten eingibt. Nach einer Plausibilitätsprüfung auf dem Client werden die Benutzerinformationen zum Server geschickt.

Animation mit Applets

Die wohl einfachste Form von Applets sind Animationen. Hier findet keine Interaktion mit dem Benutzer statt. Vielmehr dienen diese Animationen entweder der grafischen Auflockerung von Web-Seiten oder zum Abspielen von Informationssequenzen (z.B. Scrollines, Börsenticker, Filme, Sounds). Ein evidenter Vorteil von Applets hier: bis dato mussten Viewer/Player für Dateien mit Informationssequenzen getrennt vom Browser auf dem Client-System mitgeführt werden (JPEG-Viewer für JPG's, WAV-Player für WAV's). Jetzt können mit Hilfe von Applets Informationssequenzen vom Benutzer genutzt werden, ohne sich über das Format der Dateien Sorgen machen zu müssen, da der Java-Bytecode einen Viewer/Player enthalten kann.

Hier zunächst ein Beispiel für ein einfaches Animationsapplet:

```
<HTML>
<HEAD>
<TITLE> Hallo... Testwebseite </TITLE>
```

```
</HEAD>
<BODY>
wie gehts ??
<P>
<applet code = ImageLoopItem width = 80 height = 90>
<param name = nimgs value = 10>
<param name = img value = duke>
<param name = align value = right>
<param name = pause value = 100>
</applet>
</BODY>
</HTML>
```

In diesem Beispiel werden die in gezeigten GIF-Dateien übereinanderkopiert. Durch die Trägheit den menschlichen Auges sehen wir ein bewegtes Bild.

Animierte Applets müssen aber nicht zwangsweise übereinanderkopierte Bitmaps sein; der Bytecode kann ja auch direkt auf die angezeigte Bitmap wirken und diese so animieren. Ein Beispiel hierfür ist das Feuerwerk-Applet oder das Clock-Applet (http://java.sun.com).

Interessant ist das LED Sign Applet (http://java.sun.com).. Neben bloßer Parameter bietet es eine Makrosprache zur Anzeige einer Laufschrift an.

Wichtig für den Business-Bereich könnten die Börsenticker werden: Ein Applet ist für die grafische Darstellung von Charts (z.B. Aktienkurse) auf dem Client-Browser zuständig. Diesem Chart-Applet müssen dann nur reine Nutzdaten aus dem Netz übergeben werden, die vom Chart-Applet auf dem Client angezeigt werden.

Interaktion mit Java-Applets

Mit Hilfe der CGI-Programmierung kann der Web-Entwickler in eine Anwendung ein gewisses Maß an Interaktivität einbauen und somit dem Benutzer eine Möglichkeit an die Hand geben, auf Anfrage exakt auf seine Bedürfnisse zugeschnittene Informationen zu erhalten. CGI-Programme können einem Benutzer auch gestatten, eine Informationsstruktur zu ändern oder ihr etwas hinzuzufügen. Wegen seines ausführbaren Inhaltes ist mit Java ein noch höherer Grad an Interaktivität möglich.

Bei der CGI-Programmierung werden die Ressourcen des Client-Rechners nicht genutzt. An jedem Interaktionspunkt findet ein Zugriff auf den Server statt, da bei CGI keine Fachlogik mit der HTML-Seite heruntergeladen wird. So wird das Netz und der WWW-Server oft unnötig belastet und die Antwortzeit sinnlos in die Höhe getrieben. Verwendet man Java-Applets, kann ein Teil der Fachlogik auf den Client gelegt werden, so daß nur in notwendigen Fällen ein Server-Zugriff stattfindet.

Bei Applets mit reiner Benutzer-Client Interaktion können Serverzugriffe vollkommen verhindert werden. Beispielhaft soll dies anhand des Spreadsheet-Applets (und) verdeutlicht werden.

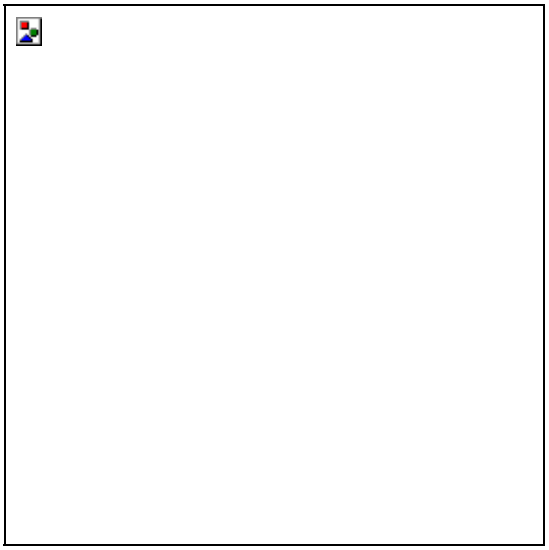


Abbildung : Client-Ressourcen werden genutzt, unnötiger Serverzugriff verhindert

```
<title>Stock Positions</title>
<h1>My Portfolio</h1>
<hr>
<applet codebase=SpreadSheet code="SpreadSheet.class" width=320 height=120>
<param name=title value="Example">
<param name=rows value="4">
<param name=cols value="3">
<param name=A1 value="v10">
<param name=A2 value="v30">
<param name=B1 value="v500">
<param name=B2 value="v1000">
<param name=C1 value="fA1*B2">
<param name=C2 value="fA2*B2">
<param name=C3 value="fC1+C2">
</applet>
<hr>
```

Abbildung : HTML-Code zu Spreadsheet-Beispiel

Beim Spreadsheetbeispiel wurde die gesamte Fachlogik in das Applet verlagert. Manchmal ist eine Interaktion zwischen Server und Client-Browser (wie im KFZ-Versicherungsbeispiel aus Kapitel 5.4) notwendig um Nutzdaten zu übertragen oder Transaktionen durchzuführen. An dieser Stelle kann das Börsentickerbeispiel weitergesponnen werden: Einem Aktienanalysten werden nicht nur die aktuellsten Kursdaten übertragen, man könnte Ihm ein zusätzliches Feature bieten, das Ihm Transaktionen, z.B. den Kauf und Verkauf von Aktien, ermöglicht.

Außerdem können durch Java-Applets nicht nur weltumspannende Multiusergames realisiert werden. Die gesamte Groupware eines Unternehmens kann basierend auf Java einheitlich umgesetzt werden.

In Anbetracht der von Oracle, Sun und anderen Firmen geplanten diskless Settopboxen kann angenommen werden, daß in naher Zukunft sogar Applikationen “gemietet” werden. Wieso soll man sich eine teure Textverarbeitung mit Tausenden ungenutzten Features kaufen, wenn man sich im Internet diese mit “Rent-an-Application” auf Zeit mieten kann.

5.3.6 Sicherheitsaspekte

Dadurch, daß fremder Code (in Form von Applets) über das Internet auf den lokalen Rechner geladen und ausgeführt wird, wurde der Ruf nach Sicherheitsmaßnahmen laut. Wer möchte schon, daß ein Applet die Daten der lokalen Festplatte unbemerkt einem unbekannten Dritten zuspielt, oder daß ein Applet gar die Festplatte formatiert?

Für diesen Zweck wurden daher Maßnahmen getroffen, die die unbemerkte Informationsübermittlung und die Schädigung des lokalen Computers unmöglich machen sollen.

Diese Sicherheit wird dadurch erreicht, daß Java-Quellcodes zuerst in Byte-Code-Anweisungen kompiliert werden,

die in dieser Form verifiziert werden können. Die Java-Anweisungen im Byte-Code ähneln anderen Befehlssätzen von Computerchips, jedoch sind sie nicht für ein bestimmtes Computersystem spezifisch und können somit auf eventuelle Schutzverletzungen geprüft werden (siehe Abbildung 14).

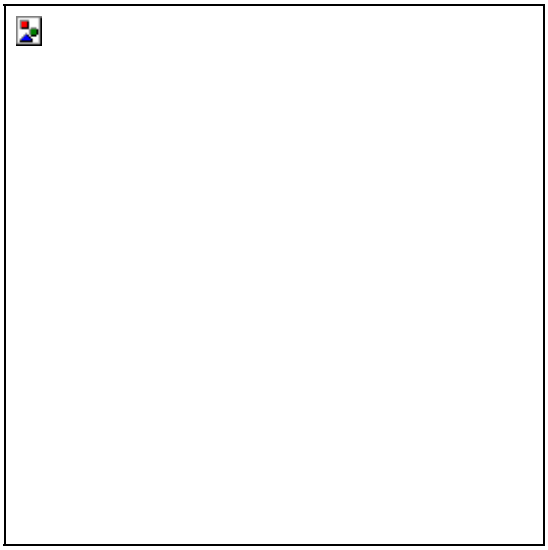


Abbildung : Der Verifizierungsprozeß für den Byte-Code bei Java.

Anders als bei herkömmlichen Programmiersprachen erfolgt der Zugriff auf Methoden und Variablen stets über ihren Namen. So läßt sich leicht verifizieren, welche Methoden tatsächlich benutzt werden. Dies ist auch notwendig, damit ein unbefugtes Herumbasteln an den Byte-Codes ausgeschlossen werden kann.

Nachdem die Applets verifiziert wurden, können sie in einer beschränkten Umgebung ablaufen. In dieser Umgebung können bestimmte gefährliche Funktionen von Applets nur ausgeführt werden, wenn ihnen das ausdrücklich gestattet wird. Aufgrund der Verifizierung ist ein Ausbrechen aus dieser Umgebung nicht möglich.

Weiterhin gilt für verifizierte Applets:

- es ist kein Operandenstack-Overflow möglich
- alle Operationen werden auf Operanden mit korrektem Typ angewendet
- es existieren keine unzulässigen Typkonvertierungen
- die Zugriffsrechte auf Felder (public, private, protected) werden eingehalten

Die Gestaltung der Zugriffsmöglichkeiten für externe Applets lassen sich in den Java-fähigen Web-Browsern einstellen. Dabei kann es folgende Möglichkeiten geben (dies variiert zwischen den Browsern):

- **None**: Die Applets haben keinen Zugriff auf das Netzwerk. Dies ist der beste Schutz, jedoch sind in diesem Modus nicht alle Applets ablauffähig.
- **Applet Host**: Hier haben die Applets nur Zugriff auf den Host, von dem sie stammen. Diese Applets können daher nicht auf die Informationen auf ihren Computern zugreifen. Dies ist auch die Standardeinstellung in der die meisten Applets ablauffähig sind.
- **Firewall**: Applets außerhalb der Firewall können nur auf Hosts, die ebenfalls außerhalb der Firewall sind, zugreifen. Dieser Modus ist natürlich nur möglich, wenn auch eine Firewall eingerichtet wurde.
- **Unrestricted**: Hier können Applets zu allen Hosts im Internet Verbindungen herstellen. Dieser Modus sollte allerdings nur benutzt werden, wenn keine Sicherheitsbedenken bezüglich des Applets vorhanden sind.

Lokal gestartete Applets (dies sind Applets im CLASSPATH!!) können natürlich auf den lokalen Rechner zugreifen. Für externe Applets kann zusätzlich der Zugriff auf die lokale Klassenbibliothek eingeschränkt werden. Dadurch wird ein Überschreiben dieser als sicher akzeptierten Klassen unterbunden.

Anzumerken bleibt, daß die Sicherheit von Java und speziell von Java-Applets durch Einsatz kryptografischer Verfahren (Public-Key-System) bei der Übertragung von Klassen und durch die eindeutige Identifizierung des Herstellers noch erhöht werden könnte.

Literaturverzeichnis

Back, Svend [Heißer Kaffee, 1996]. “Heißer Kaffee”. *c't.* Ausgabe 2 (1996): S. 138-142.

Behme, Henning [Kaffeehaus, 1995]. “Im Kaffeehaus”. *iX Multiuser-Multitasking-Magazin.* Ausgabe 7 (1995): S. 120-125.

December, John [JAVA, 1996]. *JAVA - Einführung und Überblick.* 1. Auflage. Haar bei München: Markt&Technik, 1996.

Gosling, James [White, 1995]. *The Java Language Environment A White Paper.* MountainView: Sun Microsystems, 1995.

Luckhardt, Norbert, Storm Ingo [Ballonrennen, 1996]. “Bollonrennen”. *c't.* Ausgabe 2 (1996): S. 136.

Schneider, Ute [Applets, 1996]. “Applets, schöne Applets”. *iX Multiuser-Multitasking-Magazin.* Ausgabe 5 (1996): S. 62-68.

Storm, Ingo [Verletzungsgefahr, 1996]. “Verletzungsgefahr”. *c't.* Ausgabe 5 (1996): S. 72-73.

Templ, Josef [Schwarzes Loch, 1996]. “Schwarzes Loch”. *iX Multiuser-Multitaskting-Magazin.* Ausgabe 5 (1996): S. 70-73.

Van Hoff, Arthur [Java-Applets, 1996]. *Java-Applets erstellen und nutzen: Interaktive Web-Seiten selbstgemacht*. 1. Auflage. Bonn, München, Paris u.a.: Addison-Wesley, 1996.