RRZN-Handbuch JAVA, 1. Band , 10. Auflage

(Das Inhaltsverzeichnis erstreckt sich auch über den 2. Band!)

Inhaltsverzeichnis

Kapitel 1

Einleitung und Vorwort

1.1 Vorwort zur 10. Auflage

Aufbauend auf Erfahrungen aus Veranstaltungen zur Einführung in die objektorientierte Programmierung mit Java an der Universität Hannover und der Hochschule für Technik Stuttgart entstand in den letzten Jahren dieses Buch.

Inhalt und Aufbau dieses Buches beruhen daher in weiten Teilen auf den Erfahrungen aus diesen Kursen. Die Themen werden dabei in diesem Buch ausführlicher und tiefergehender dargestellt, als dies in einer Vorlesung möglich ist.

Die ersten Kapitel geben einen Überblick über die Programmiersprache Java und zeigen die Schritte bis zum ersten lauffähigen Java-Programm "Hallo Java". Format, Zeichensatz und Grundelemente wie Datentypen, Schleifen-Konstrukte und Variablen gehören zu jeder Programmiersprache und müssen auch in einem Einführungsbuch zu Java dargestellt werden.

Die zwei Kapitel "Objektorientierte Programmierung" und "Vererbung" widmen sich ausführlich den Grundlagen der objektorientierten Programmierung. Dazu gehören auch tiefer gehende Themen der OOP wie abstrakte Klassen, Interfaces und verschachtelte Klassen.

Aufbauend auf den Grundlagen der objektorientierten Programmierung werden Java-Anwendungen mit Kommandozeilenoberfläche, als Applets, sowie mit Grafischen Benutzeroberflächen gestaltet.

Der zweite Band vertieft die Kenntnisse des ersten Bandes durch die Themen Swing, Exceptions, Grafik, Streams, Netzwerkprogrammierung in den jeweiligen Kapiteln und dabei die dafür notwendigen Programmiertechniken und Java-Bibliotheken (API, Application Programming Interface) vorstellt.

Weitergehende und speziellere Java-Anwendungen, wie Datenbankzugriffe mit JDBC, RMI o.ä, werden in dieser Einführung ebenfalls berücksichtigt. Ausdrücklich nicht behandelt werden Techniken der Software-Entwicklung.

In den letzten 4 Jahren wurde das Buch dreimal durchgehend überarbeitet, in der 7. Auflage wurden die Inhalte an die neuen Sprachelemente und Bibliotheken der Java-Versionen 5 und 6 angepasst. Diese Änderungen können sicherlich zu recht als die weitreichendsten Modifikationen angesehen werden, die Java nach der ersten stabilen Version im Jahr 1998 erfahren hat. Von diesen Änderungen profitierten sowohl Anfänger als auch Experten. Im Zuge der Überarbeitung

wurde die Behandlung einiger anderer Themen gekürzt. Insbesondere wurde im Graphikbereich auf die Behandlung der AWT-Dialogelemente verzichtet und stattdessen die Programmierung mit Swing-Dialogelementen behandelt.

Dafür wurden die Bereiche JDBC, JSP/Servlets und RMI neu eingeführt, da sie an vielen Hochschulen bereits im Rahmen der Anfängerausbildung behandelt werden. Um Programmieranfängern den Umgang mit Werkzeugen zu erleichtern, gibt es ergänzend zu diesem Skript unter `http://www.hft-stuttgart.de/Informatik/Skripte` Mitschnitte von der Benutzung dieser Werkzeuge. Dort finden sich auch Beispiele, die aus Platzgründen nicht abgedruckt werden konnten.

Die in der 7. Auflage vorgenommenen Änderungen hatten die Popularität des Skriptes offenbar erheblich gesteigert, daher wurde schon nach einem knappen Jahr die nächste Auflage fällig. Außer vielen Korrekturen—die meisten von ihnen von aufmerksamen Leserinnen und Lesern angeregt—wurden inhaltlich einige Ergänzungen vorgenommen, um das Buch an den neuen Sprachstandard von Java 7 anzupassen. Hierzu gehören unter anderem die Beschreibungen von Closures, einige Vereinfachungen bei generischen Typen, Automatischen Ressourcen und Modulen[1].

Wegen der zuletzt wenig zufriedenstellenden Ergebnisse des von FrameMaker erzeugten Layouts wurde die 8. Auflage in LaTeX erstellt. Eine Rohfassung wurde mit Hilfe einiger selbstgeschriebener Java-Programme aus dem SGML-Export der FrameMaker-Dateien erzeugt. Diese Rohfassung wurde dann an etlichen Stellen, insbesondere beim Tabellensatz und bei eingefügten Abbildungen, manuell nachbearbeitet, um das endgültige Layout zu generieren. In der 9. und 10. Auflage wurden dann die mannigfachen Tippfehler korrigiert, die sich beim Übertragen eingeschlichen hatten, außerdem wurden noch Kleinigkeiten am Layout geändert und sogenannte Abkommen im Bereich Objekte eingeführt.

Nach der Übernahme der Firma Sun durch Oracle haben sich außerdem viele HTTP-Links geändert. Da diese Änderungen noch nicht vollständig abgeschlossen sind, tragen alle Links ins Netz einen zusätzlichen numerischen Index, z. B. der Form `http://www.oracle.com/technetwork/java/index.html`$_1$. Über diesen Index lässt sich später auf der Webpräsenz des Buches ein möglicherweise aktualisierter Link finden. Das gleiche gilt für im Buch dargestellte Programme, hier werden durch die geänderten Links auch ggf. neuere Programmversionen berücksichtigt.

1.2 Voraussetzungen

Das Buch wendet sich an alle, die die Programmierung mit Java erlernen wollen, wobei insbesondere auch denjenigen der Einstieg in die Java-Programmierung erleichtert werden soll, die bisher noch keine oder nur sehr wenig Erfahrung in der objektorientierten Programmierung haben. Die Grundlagen der objektorientierten Programmierung werden ausführlich und unter unterschiedlichen Gesichtspunkten dargestellt.

Das Buch sollte außer als Begleitmaterial zu Kursen auch zum selbstständigen Erlernen der Programmiersprache Java und zum Vertiefen bestimmter Themen der Java-Programmierung geeignet sein.

Vorausgesetzt werden in diesem Buch ausreichende Kenntnisse über das Betriebssystem des eigenen Rechners. Wer Java-Programme entwickeln will, sollte

[1] Viele Features werden erst mit Java 8 Mitte 2012 erscheinen

im Umgang mit Dateien, Dateisystemen und einem Editor auf dem Rechner vertraut sein und möglichst auch Erfahrungen mit der Installation von Software haben. Kenntnisse über das Internet und das WWW sollten ebenfalls vorhanden sein.

Erfahrungen mit einer anderen Programmiersprache, wie C, C++, Fortran, Basic, o. ä. erleichtern zwar den Einstieg in die Java-Programmierung, werden aber nicht vorausgesetzt. Aus diesem Grunde werden etliche Begriffe auch peu-à-peu eingeführt, so dass dem Anfänger am Anfang genug Informationen zur Verfügung stehen, um die Begriffe nutzen zu können.

1.3 Layoutkonventionen

Alle längeren Programmbeispiele sind zweispaltig in der Schrift `LuxiMono` dargestellt, wobei die Spaltenbreite auf 40 Zeichen begrenzt ist. Längere Zeilen werden umgebrochen, der Umbruch wird durch das Symbol ✂ dargestellt. Programmfragmente werden im Gegensatz zum normalen Fließtext nicht durch die übliche Texttrennung umgebrochen, sondern einfach dort, wo der Platz in der Zeile zuende ist. An diser Stelle wird dann das Zeichen » eingefügt. In beiden Fällen muss beim Abtippen eine solche Zeilen ohne Leerzeichen mit der Folgezeile zusammengefügt werden.

```
1 public class Test {                          ✂ ng outputFile, int indent) {
2   int x,y;                                   8     // Ein Kommentar, in dieser Methode nu✂
3   public void foo() {                        ✂ r als Beispiel für Kommentare
4     String x = "Hello World";                9   }
5     System.out.println(x);                   10 }
6   }                                          11
7   void workMethod(String description, Stri✂
```

Programm 1-1: Beispiel für Programmtext

Bei der Beschreibung von Syntax-Elementen werden die von der Programmiersprache fest vorgegebene Worte, z. B. **`public`** in `LuxiMono` **`fett`** dargestellt. Bestandteile, die als Platzhalter für veränderbare Elemente stehen, z. B. `Test` sind in `LuxiMono` dargestellt, Zeichenketten und Kommentare, z. B. *`"Hello World"`* in `LuxiMono` *`kursiv`*.

Im Beispiel *`modifikator`* **`class`** *`name`*`{}` sind das Schlüsselwort **`class`** und die geschweiften Klammern `{ }` fest vorgegebene Bestandteile der Beschreibung, *`name`* sowie *`modifikator`* stehen für Elemente, die bei der Anwendung durch konkrete Elemente zu ersetzen sind.

Runde Klammern hinter einem Namen weisen immer auf eine Methode hin. Parameter und Modifikatoren, die beim Aufruf der Methode ggf. anzugeben sind, werden im Fließtext normalerweise weggelassen.

Beispiel: Die Methode `main()` wird normalerweise durch **`public % static void`** `main(String[] args)` deklariert. Im Fließtext wird aber nur die Schreibweise `main()` ohne Parameter zur Bezeichnung der Methode verwendet.

Die Beschreibungen zu Pfaden und Verzeichnissen und Umgebungsvariablen sind durchgängig in der Notation für Windows-Systeme dargestellt. Für Linux, Solaris und andere unixartige Syteme müssen diese Angaben übersetzt werden, indem die Trennzeichen `';'` und `'\'` der Windows-Variable in die Trennzeichen

`':'` und `'/'` der Unix-Variable geändert werden. Die Deklaration der Umgebungsvariablen `CLASSPATH=.;c:\java` wird dann zu `CLASSPATH=.:/opt/java`.

Bei den ausführbaren Programmen sind die meisten Kommandos für beide Systeme gleich, da die Endung `.exe` beim Programmaufruf nicht mitgeschrieben werden muss. Es heißt daher immer `javac xyz.java` und nicht `javac.exe xyz.java`

1.4 Dokumenten-Erstellung und Online-Version

Bis zur 7. Auflage wurde das Buch mit dem Programm "Adobe FrameMaker" erstellt. Nachdem dieses Produkt jedoch beim Hersteller immer weniger im Zentrum des Interesses steht, musste eine Entscheidung getroffen werden, mit welchem Satzsystem das Buch künftig gedruckt werden soll. Die Wahl fiel auf das System LaTeX, weil es einerseits ein hervorragendes Layout liefert und andererseits auch extrem flexibel ist. So konnten alle Programm-Beispiele automatisch aus dem Quelltext extrahiert, gemäß den im vorigen Abschnitt angegebenen Konventionen formatiert und dann wieder in das Buch aufgenommen werden. Als Nebenprodukt ergab sich eine nach Kapiteln sortierte Zusammenstellung aller Java-Quelltexte.

Um die Druckkosten niedrig zu halten, wurden etliche Beispiele im Abdruck auf das absolute Minimum reduziert. Dem sind in manchen Fällen auch Teil zum Opfer gefallen, die zwar nicht für das Verstehen, jedoch für das Übersetzen der Programme wichtig sind. Falls abgetippte Programme nicht übersetzbar sind, sollten Sie die vollständigen Quelltexte der Beispiele unter der URL `http://www.hft-stuttgart.de/Informatik/Skripte` anschauen.

Diese Seite verweist auf das Moodle-System der HFT-Stuttgart, wo Sie nach einem Klick auf den Button "Als Gast anmelden" die Quelltexte herunterladen können. Dort finden Sie auch weiterführende Informationen., z. B. das vollständige Inhaltsverzeichnis sowie den Index zu beiden Bänden (nach Fertigstellung von Band 2).

Abbildung 1.1: `Moodle-Fenster`

Naturgemäß kann ein gedrucktes Buch, insbesondere als Lehrbuch, nur einen kleinen Teil der verfügbaren Informationen enthalten. Die vollständigen Informationen finden sich in der Online-Dokumentation der Firma Oracle. Die Details zu allen Klassen und ihren Methoden finden Sie unter: `http://java.sun.com/javase/6/docs/api`[2]

Beispiele, wie die Klassen benutzt werden können (sog. HOWTOs) finden Sie unter `http://java.sun.com/docs/books/tutorial`[3].

Alle drei Webpräsenzen sollten Ihr ständiger Begleiter sein. Da nicht ganz auszuschließen ist, dass die angegebenen Adressen sich in der nächsten Zeit noch ändern, sind diese Adressen durchnummeriert, so dass Sie notfalls über Moodle-Seite der HFT Stuttgart die aktuelle Version der Adresse finden können.

1.5 Dank

Danken möchten wir allen denjenigen, die z. T. mehrere Entwürfe dieses Buches gelesen, konstruktive und detaillierte Kritik geäußert, viele Ideen eingebracht und Fehler verhindert haben: Herr Christof Graß-De Iulius, Herr Bernd Eggink, Herr Klaus Fleck, Herr Ralph Hagl, Herr Andreas Kopecki, Herr Prof. Dr. Claus König, Herr Dirk Krämer, Herr Prof. Dr. Hermann Matthies, Frau Prof. Dr. Petra Scheffler, Frau Astrid Tessmer und Herr Dr. Gerd Wegener.

Für die 7. Auflage wurden hilfreiche Vorschläge von vielen Kollegen gemacht, u.a. von Frau Prof. Gabriele Bannert, Herrn Dr. Günther Bauer, Herrn Prof. Dr.-Ing. Jörg Berdux, Herrn Prof. Dr. Marcus Deininger, Herrn Dr.-Ing. Udo Gnasa, Herrn Prof. Dr.-Ing. Helmut Hermann, Frau Dr. Monika Meiler und Herrn Prof. Dr. Alexander Rausch. Die Druckfassung wurde von Frau Anja Aue korrekturgelesen.

Viele Fehler der 7. Auflage wurden von Frau Prof. Gabriele Bannert und Herrn Alfred Lindner aufgelistet. Bei der Umstellung von FrameMaker auf LaTeX hat Frau Barbara Sippli Unterstützung geleistet, insbesondere bei der Re-Integration der Zeichnungen und Abbildungen sowie bei der Zusammenstellung der Querverweise. Für die 8. Auflage haben Herr Prof. Detlef Lobenstein und Herrn Wolfgang Husmann viele Hinweise gegeben.

Herausgeber dieses Buches ist Wilhelm Noack, Initiator und Koordinator der vom RRZN geleiteten Arbeitsgemeinschaft staatlicher Hochschulen auf dem Gebiet der IT-Skripte. Ihm gilt unser besonderer Dank für stete Ratschläge und die vorzügliche Begleitung des Entstehungsprozesses.

1.6 Autor

Bis zur 4. Auflage wurde dieses Buch von Dr. Thomas Kröckertskothen vom RRZN Hannover betreut, die 5. und 6. Auflage waren unveränderte Nachdrucke.

Ab der 7. Auflage erfolgte die Überarbeitung durch Dr. Peter Heusch von der HFT Stuttgart (`Peter.Heusch@hft-stuttgart.de`).

Er trägt die Verantwortung für den Inhalt und alle Fehler, seien Sie alt und unentdeckt oder neu eingefügt. Für Vorschläge, Anregungen und Kritik ist er jederzeit dankbar.

Kapitel 2

Java im Überblick

Seit die Programmiersprache Java im März 1995 vorgestellt wurde, hat sie wie kaum eine andere Sprache die Entwicklung der Informatik beeinflusst. Anfangs als Spielzeug belächelt und bestenfalls für Animationen im Web-Browser benutzt, ist Java heute eine der meistverbreiteten Programmiersprachen. Aktuell ist nach dem weitverbreiteten TIOBE-Index `http://www.tiobe.com` Java mit knapp 19% die meistverbreitete Programmiersprache vor C und C++.

Mit der Java nachempfundenen Programmiersprache C# haben diese drei Sprachen einen Marktanteil von über 50%, wobei Java seit 10 Jahren "Marktführer" ist. Dieses Kapitel gibt Ihnen einen kleinen Überblick über die Programmiersprache, die Historie und die grundlegende Technik, die sich "unter der Haube" befindet.

2.1 Insel, Kaffee und Programmiersprache

Indonesien

Bevor es auf den folgenden Seiten nur noch um die Programmierung geht, soll zunächst noch einmal ein Blick aus der Programmierwelt heraus in die reale Welt erfolgen: Java (132.186 km^2) ist nach Sumatra (473.481km^2) die zweitgrößte Insel Indonesiens.

Sun Microsystems

In den USA ist Java der umgangssprachliche Ausdruck für eine Tasse Kaffee. Die Tasse Kaffee gilt auch als der "Betriebsstoff" der Java-Programmierer, die dampfende Java-Tasse ist das offizielle Java-Logo der Firma Sun Microsystems.

Seit 1994 ist Java auch die Bezeichnung für eine neue Programmiersprache. Die Programmiersprache Java ist unmittelbar mit dem Namen der Firma Sun Microsystems verbunden, in deren Reihen 1991 erste Entwicklungen zu Java begannen. Das Projekt lief damals unter der Bezeichnung Project Green. Ziel war es, eine einfache Programmiersprache mit grafischer Benutzeroberfläche zu entwickeln, die in Geräten der Haushalts- und Unterhaltungselektronik eingesetzt werden sollte.

Im Gegensatz zu "normalen" Computern, bei denen der Benutzer regelmäßige Programmabstürze akzeptiert, sollten diese Geräte mehrere Eigenschaften haben:

- Programme müssen in einem maschinenunabhängigen Format gespeichert werden, da Geräte dessselben Herstellers mit unterschiedlichen Prozessoren bestückt werden.

- Software muss im Betrieb zuverlässig sein, Abstürze, Speicherlecks und die gegenseitige Störung (versehentlich oder absichtlich) von Programmen sowie das Ausspionieren der Daten anderer Programme müssen ausgeschlossen sein.
- Während des Betriebs muss die Software aktualisiert werden können, z. B. um neue Verschlüsselungsverfahren im Bereich des Bezahlfernsehens auf die Satellitenempfänger herunterladen zu können.
- Die Benutzung soll durch eine grafische Benutzeroberfläche vereinfacht werden.

Das aus diesem Projekt entstandene Gerät (ein PDA-ähnliches Gerät) kam nie auf den Markt, da es auf dem Markt zu diesem Zeitpunkt noch keinen Bedarf an solchen Geräten gab. Auch andere Ansätze, z. B. für Satellitenempfänger, waren nicht erfolgreich. Mit dem Aufkommen des World Wide Web ergab sich jedoch für Java eine zusätzliche Einsatzmöglichkeit. Die Web-Seiten jener Zeit waren meist relativ rudimentär und dynamische Inhalte wurden über sogenannte CGI-Seiten erzeugt. Auf einer solchen Seite klickt der Benutzer etwas an (z. B. ein Button oder eine Landkarte), diese Information wird zum Server gesendet, der daraufhin eine neue Seite erzeugt und zum Client zurücksendet.

CGI: Common Gateway Interface

Theoretisch ließen sich die Browser zwar durch neue Module (Plugins) erweitern. In der Praxis geschah dies jedoch nur selten, da der Code für ein solches Modul aus dem Netz heruntergeladen wurde (Sicherheitsrisiko), unter Umständen nicht zuverlässig war (Absturzrisiko) und für alle Client-Systeme separat vorgehalten werden musste (Maschinenabhängigkeit). Mit Java konnten alle diese Probleme auf einen Schlag gelöst werden. Das einzige Plugin, welches noch nachgeladen werden musste, war das Plugin für Java selbst. Danach waren nahezu beliebige aktive Browserinhalte darstellbar.

Netscape Mozilla

Netscape war der erste Browser-Hersteller, der Java unterstützte. Der 1995 herausgegebene Netscape-Navigator 2.0 unterstützte erstmalig Java, das damals noch in der Beta-Version vorlag. Die erste Java-Version 1.0 wurde 1996 veröffentlicht, sie wurde sehr schnell auch von anderen Herstellern in ihre Browser integriert.

2.2 Programmiersprache Java

Die Programmiersprache Java wurde ohne falsche Rücksicht auf vorhandene Programmiersprachen entworfen. Die neue Sprache sollte möglichst einfach sein, bewährte Merkmale vorhandener Sprachen beinhalten, dabei aber auf fehleranfällige und unnötig komplexe Bestandteile verzichten. Als Ziele wurden definiert: Einfachheit, Objektorientierung, Netzwerkfähigkeit, Bytecoding, Robustheit, Sicherheit, Portabilität, Geschwindigkeit, Parallelität und Dynamisches Binden.

Einfachheit

Java soll einfach sein. Um dieses Ziel zu erreichen, besitzt die Sprache wenige grundlegende Sprachkonstrukte. Im Gegensatz zu COBOL, das mehr als 250 Schlüsselworte enthält, hat Java nur ca. 50 reservierte Worte. Bewährte Konstrukte anderer objektorientierter Sprachen wie C++ und Smalltalk wurden

übernommen, auf komplexe und fehleranfällige Sprachkonstrukte wurde verzichtet.

Die aus C bekannten und häufig zu schwer diagnostizierbaren Fehlern führenden Zeiger sind in einer modernen objektorientierten Programmiersprache überflüssig und wurden durch Referenzen ersetzt.

Die Syntax von Java ist an die Sprachen C und C++ angelehnt, dadurch wird vielen Programmierern der Umstieg auf Java erleichtert. Trotz dieser Einfachheit in der Struktur ist Java vollständig und sehr leistungsfähig. Es sollte jedoch erwähnt werden, dass sich die einfache Struktur von Java erst bei voller Kenntnis der objektorientierten Konzepte erschließt und dem Programmieranfänger durchaus komplex erscheinen kann.

Objektorientierung

Java ist objektorientiert. Es enthält keine Kompromisse in Richtung prozeduraler Sprachelemente. Außer den wenigen Grunddatentypen für Zahlen, Zeichen und Wahrheitswerte sind alle anderen Daten in Java Objekte. Es gibt keine Unterprogramme oder Prozeduren in Java, sondern nur noch Methoden, die zwingend in Klassen definiert sind.

Objektorientierte Programmierung (OOP) ist ein modernes Programmierparadigma und für komplexe Anwendungen ausgesprochen hilfreich. Das klare objektorientierte Konzept von Java ist sicher auch einer der Gründe dafür, dass Java zunehmend in der Informatik-Ausbildung eingesetzt wird.

Netzwerkfähigkeit

Java wurde von Beginn an mit dem Ziel entworfen, verteilte Anwendungen zu unterstützen, also Anwendungen, die im Netzwerk verteilte Ressourcen nutzen. Zu Java gehören standardmäßig Programmierschnittstellen für die Datenkommunikation im Internet, mit denen sich vollständige Client-Server-Applikationen besonders elegant realisieren lassen.

Schon ganz früh wurden verteilte Anwendungen durch RMI (Remote Method Invocation) unterstützt, mit RMI können Java-Programme auf Objekte und Methoden zugreifen, die auf anderen Rechnern liegen, ohne sich um die Details der Kommunikation kümmern zu müssen. Dies ist (mit CORBA[1]) sogar mit Programmen auf der anderen Seite möglich, die gar nicht in Java geschrieben wurden.

Bytecoding

Der Compiler erzeugt einen maschinenunabhängigen Java-Bytecode (siehe dazu auch den Abschnitt Java Bytecode und die "Virtuelle Maschine"). Um Java-Bytecode auszuführen, ist ein Java-Interpreter notwendig, der den Bytecode interpretiert. Java-Bytecode

Die Maschinenunabhängigkeit des Bytecodes ist Vor- und Nachteil zugleich. Vorteil, weil Programme auch in der übersetzten Form auf anderen Maschinen ohne Änderung ausgeführt werden können. Nachteil, weil so die Möglichkeiten spezieller Prozessoren nicht ausgenutzt werden können. Es gibt jedoch auch

[1] CORBA: Abk. f. Common Object Request Broker Architecture, System zur Kommunikation zwischen verteilten Programmen im Netz.

Compiler, welche direkt ausführbaren Maschinencode erzeugen, zum Beispiel der GNU-Java-Compiler GCJ oder der Java-Compiler für das Android-System. Das zuletzt genannte System hat nicht zuletzt aufgrund der Unterstützung von Java in nur 3 Jahren aus dem nichts Platz 1 auf der Liste beliebtester Smartphone-Plattformen erreicht.

Robustheit

Java-Programme sind robust. Getestete und für fehlerfrei befundene Programme sollen fehlerfrei im Anwendungsbetrieb eingesetzt werden können. Der aus anderen Programmiersprachen bekannte Effekt, dass ein Programm viele Male korrekt abläuft und dann plötzlich abstürzt, z. B. weil Daten aus falsch belegten Speicherbereichen gelesen oder Grenzen für Datenbereiche überschritten werden, soll in Java vermieden werden. Konzepte und Maßnahmen für robuste Programme in Java sind u. a.: strenge Regeln zur Einhaltung der Konsistenz von Datentypen bereits bei der Übersetzung, keine Programmierelemente mit direktem Zugriff auf den Speicher, eine Speicherverwaltung durch das Java-System, strenge Überprüfung der Grenzen von Datenstrukturen wie Zeichenfolgen, Arrays und Vektoren.

Sicherheit

Java-Programme sollen für verteilte Anwendungen im Internet einsetzbar sein. Damit waren sofort auch entsprechend hohe Anforderungen an die Sicherheit von Java-Programmen verbunden. Das Sicherheitskonzept in den ersten Java-Versionen war das so genannte Sandkastenmodell. Java-Programme, die auf dem lokalen Rechner ausgeführt werden, haben vollen Zugriff auf alle Ressourcen des lokalen Rechners und können beispielsweise alle Dateien und Verzeichnisse anlegen, lesen, löschen usw. Programme, die über das Internet geladen und ausgeführt werden, hierzu zählen i.d.R. alle Applets, werden als nicht vertrauenswürdig eingestuft und haben nur sehr eingeschränkte Zugriffsmöglichkeiten. Im Sandkastenmodell heißt dies: Applets laufen in einem Sandkasten mit sehr restriktiven Zugriffsmöglichkeiten auf die Ressourcen des ausführenden Rechners.

Um die Zugriffsmöglichkeiten zu erweitern, kann das Applet eine digitale Signatur erhalten. Wird die digitale Signatur akzeptiert, erhält das signierte Applet alle Zugriffsmöglichkeiten eines lokalen Programms. Für besondere Anforderungen gibt es die Security Manager. Mit einem Security Manager verknüpft sind Sicherheitsregeln für die Zugriffe auf Dateien, Verzeichnisss und Netzverbindungen. Die mit dem Security Manager verfügbaren Tools ermöglichen es, die Sicherheitsregeln für alle Programme den jeweiligen Sicherheitsanforderungen anzupassen. Die Standardeinstellungen des Security-Managers erlauben lokalen Applikationen den vollen Zugriff auf die Ressourcen des lokalen Rechners, Applets haben die restriktiven Einschränkungen des ursprünglichen Sandkastenmodells.

Portabilität

Java hat den Anspruch, dass ein Programm als Quelltext auf jeder Architektur übersetzbar und als Bytecode auf jeder Rechner-Architektur ausführbar ist und

dass auch Resultate identisch sind. Ein unter Windows XP oder Windows 7 erstelltes Java-Programm kann ohne weitere Änderungen im Datennetz verteilt und unter Solaris, Linux, MacOS, ja sogar unter Großrechnersystemen wie z/OS oder BS2000 übersetzt und ausgeführt werden.

Diese Architekturunabhängigkeit von Java war ursprünglich eingeführt worden, um Applets über das Internet verteilen zu können. Mittlerweile ist die Architekturunabhängigkeit von Java auch ein großer Vorteil bei der Entwicklung von großen Anwender-Projekten. Einmal entwickelt, können die Anwendungen ohne weiteren Anpassungsaufwand auf allen Plattformen eingesetzt werden.

Die Sprachdefinition von Java erfüllt den Anspruch, keinerlei systemabhängige Elemente zu enthalten. So sind u. a. die Größen aller Grunddatentypen in Java systemunabhängig in der Sprache festgeschrieben, gleiches gilt für viele der in der Laufzeitumgebung verfügbaren Klassen.

Geschwindigkeit

Wird unter Geschwindigkeit nur Geschwindigkeit bei der Ausführung von Programmen verstanden, so hat Java sicherlich gewisse Nachteile. Die Ausführung von architekturneutralem Bytecode kann niemals schneller sein als die Ausführung optimaler C- oder Assembler-Programme in Binärcode.

Es gibt jedoch Methoden der Programmbeschleunigung durch Hard- oder Software. Die Beschleunigung durch Hardware nutzt einen Prozessor, der alle oder einzelne Bytecode-Befehle als Maschinencode kennt (zum Beispiel der Atmel AVR32). Bei der Beschleunigung durch Software übersetzt die JVM oft benötigten Bytecode in Maschinencode der Zielmaschine (sog. JIT-Compiler), der dann entsprechend schnell bearbeitet wird.

JVM: Java Virtual Machine
JIT: Just in Time

Viel wichtiger als die Optimierung auf der Basis der Maschinenbefehle ist jedoch die Benutzung guter Algorithmen. Ein guter Algorithmus wird auf Dauer schneller als ein schlechter Algorithmus sein, selbst wenn der schlechte Algorithmus optimal implementiert ist. Insofern wird Java mit seiner großen Zahl mitgelieferter guter Algorithmen gerade für Anfänger häufig schnellere Programme liefern als selbst erstellte Verfahren in anderen Programmiersprachen.

Außerdem ist für viele GUI-Programme wegen der grafischen Benutzeroberfläche die Ausführungsgeschwindigkeit des Programmcodes kaum von Bedeutung, da die Benutzeraktivitäten die Gesamtausführungszeit bestimmen. Wird darüberhinaus unter Performanz die Gesamtzeit der Programmentwicklung von der Erstellung bis zur fertigen Laufzeitversion betrachtet, werden die Vorteile von Java noch größer. Durch Einfachheit, Objektorientierung und Robustheit kann bei Java die Entwicklungszeit von Programmen erheblich verkürzt werden, dies ist für den praktischen Einsatz meist wichtiger als eine um wenige Prozent höhere Geschwindigkeit.

GUI: Graphical User Interface

Nebenläufigkeit

Moderne Prozessoren haben die Fähigkeit, mehrere Programmteile parallel ablaufen zu lassen, zum Zeitpunkt der Drucklegung dieses Buches (September 2010) haben "normale" PCs schon 6 Recheneinheiten. Die Programmteile laufen in Threads (Ausführungsfäden) ab. Parallelisierung ist eine wesentliche Voraussetzung für Animationen und Netzwerk-Applikationen. Denn während die Animation bunt blinkend auf dem Bildschirm läuft oder die Netzwerk-Applikation

auf Daten wartet, sollte das System nicht für andere Tätigkeiten blockiert sein. Java ist von Beginn an mit dem Ziel entwickelt worden, dies zu vereinfachen. So sind alle Methoden in Java standardmäßig auch in nebenläufigen Programmteilen - in Threads - ausführbar. Die Programmierung von Threads wird in Java durch verschiedene Konzepte, Klassen und Methoden unterstützt. Das Thema wird in Band 2 im Kapitel 22 "Threads" gesondert behandelt.

Dynamisches Binden

Während viele andere Programmiersprachen Anwendungen beim Start als monolithischen Block in den Speicher laden, entscheidet Java zur Laufzeit, welcher Bytecode geladen werden soll. Eine Klasse wird von Java erst dann geladen, wenn sie auch tatsächlich vom Programm benötigt wird. Die Laufzeit-Typinformation ermöglicht es mit geeigneten Werkzeugen, den Typ eines Objektes und die für das Objekt verfügbaren Methoden zur Laufzeit zu bestimmen. Mit der Einführung des Konzepts der Reflection sind diese Informationen auch für Anwendungsprogrammierer zugänglich.

2.3 Java-Bytecode und die Virtuelle Maschine

Die im vorherigen Abschnitt genannten Eigenschaften architekturneutral, portabel und verteilt lassen sich auch zusammenfassen als:

WRITE ONCE - RUN EVERYWHERE.

JVM: Java Virtual Machine

Einmal geschriebene Java-Programme sind überall ausführbar. Realisiert wird dieser Anspruch durch den Java-Bytecode und die virtuelle Java-Maschine (Java-VM oder kurz auch nur VM).

javac: Java-Compiler

Java-Bytecode wird vom Java-Compiler erzeugt und in Dateien mit der Endung `.class` gespeichert. Für die Programmierung der Klasse XYZ bedeutet dies, dass zunächst das Java-Programm als Quelltext im Editor erstellt und in der Quelltextdatei XYZ `.java` gespeichert wird. Die Quelltext-Datei übergeben Sie dem javac bzw. javac.exe, und dieser erzeugt dann die Datei mit dem Bytecode. Java-Bytecode-Dateien können Applets oder Applikationen enthalten. Jede Bytecode-Datei enthält eine übersetzte Klasse. Sowohl der Name der Quelltextdatei als auch der Name der Bytecode-Datei müssen mit dem Klassennamen identisch sein. Dies gilt sogar in Bezug auf die Groß/Kleinschreibung!

Die Java-VM sorgt für die Ausführung von Java-Bytecode auf dem Rechner. Die Java-VM kann als Schnittstelle zwischen dem architekturneutralen Java-Bytecode und der spezifischen Hard- und Software des ausführenden Systems angesehen werden.

JDK: Java Development KIT

JRE: Java Runtime Environment

Die Java-VM muss schon auf dem Rechnersystem vorhanden sein, sie wird i. d. R. als Software installiert. Im Kapitel "Programmierwerkzeuge" wird beschrieben, wie die Java-Entwicklungsumgebung (JDK) installiert wird, diese enthält u.a. eine JVM. Ein Rechner, der nur zur Ausführung von Java-Programmen eingesetzt wird, benötigt dagegen nur die Laufzeitumgebung JRE.

Die Java-VM führt dabei folgende Schritte durch:

- Die Applikation wird, wenn sie signiert ist, auf korrekte Signatur geprüft. Die Signatur vermeidet zwar leider nicht, dass Programme fehlerhaft sind, allerdings ist im Fall der Fälle zumindest der Urheber leicht festzustellen.

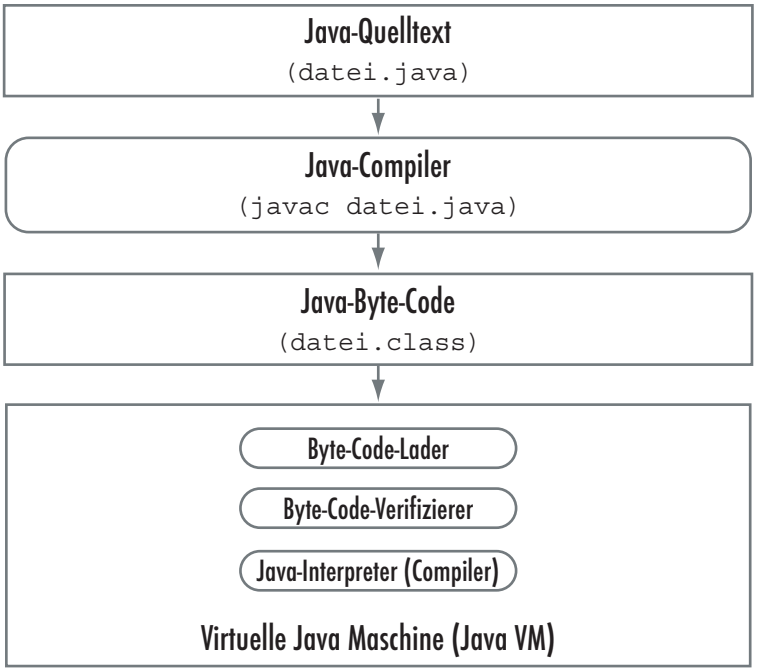

Abbildung 2.1: Virtuelle Java Maschine

- Der Bytecode wird geladen (Bytecode-Loader). Dabei werden weitere im Programm benötigte Klassen ggf. aus Bibliotheken (Klassenbibliotheken, APIs) nachgeladen.

API: Application Programming Interface

- Der Bytecode wird auf Verstöße gegen die Java-Sprachregeln untersucht (Bytecode-Verifizierung)

- Abschließend wird der Bytecode interpretiert und als maschinenspezifischer Code auf dem System ausgeführt.

WWW-Browser, die Applets ausführen sollen, müssen Zugriff auf eine Java-VM haben, Moderne Browser enthalten i. d. R. eine eigene Java-VM, oder sie benutzen das von Oracle erhältliche Plugin.

2.4 Anwendungsbereiche und Einsatzgebiete

Java ist eine Programmiersprache für allgemeine Anwendungen (general purpose language") und für alle Aufgaben geeignet. Mit Java können Sie alle Probleme lösen, für die auch andere Programmiersprachen wie C, C++, Fortran oder Pascal einsetzbar sind. Ein Vorteil beim Einsatz von Java sind die umfangreichen Programm-Bibliotheken für eine Vielzahl von Anwendungsbereichen.

In den Anfangszeiten war ein Hauptanwendungsbereich von Java die Erstellung kleiner interaktiver Komponenten für WWW-Seiten (Applets). Dort ist

Java inzwischen durch Techniken wie Flash oder Ajax[2] abgelöst worden. Dafür gewinnt Java in fast allen Gebieten moderner Software-Entwicklung enorm an Bedeutung. Einige typische Anwendungsbereiche, in denen Java heute eingesetzt wird, werden im Folgenden beschrieben.

- Client-Server-Software, d. h. Anwendungsprogramme mit Benutzeroberfläche für die Kommunikation zwischen einem Benutzerarbeitsplatz und einem Server im Datennetz, lässt sich bereits mit den im Java-Standard-System enthaltenen Methoden erstellen. Dabei übernimmt der Client die Darstellung, während der Server die Datenhaltung (z. B. in einer Datenbank mit JDBC[3]) erledigt. Diese Technik wird im Kapitel "Netzwerkprogrammierung" behandelt. Mit erweiterten Möglichkeiten, z. B. aus der JEE , lassen sich vollständige Server-Applikationen in Java realisieren.

  JEE: Java Enterprise Edition

  Mit den Konzepten Servlet und JSP (Java Server Pages) bietet Java Werkzeuge zur Erstellung dynamischer WWW-Anwendungen. Ein Servlet ist ein "Stückchen" Java-Code, der vom Webserver (z. B. Apache) ausgeführt wird, um eine vollständige HTML-Seite zu erzeugen. Eine Java Server Page ist dagegen eine HTML-Seite, die eingebetteten Java-Code enthält. Der Vorteil von JSP gegenüber Servlets liegt darin, dass man JSP-Dateien mit HTML-Werkzeugen bearbeiten kann, während Servlets alles (auch die festen Teile einer HTML-Seite) durch die Ausgabe von Java-Methoden erzeugen müssen. Demgegenüber besitzen Servlets den Vorteil, im Webserver selbst keinen eingebauten Compiler zu benötigen und den Benutzer am Browser nicht mit irgendwelchen Fehlermeldungen aufgrund nicht übersetzbaren Java-Codes zu verwirren. Servlets und JSP bieten gegenüber Verfahren wie CGI höhere Sicherheit und Geschwindigkeit. Auch beim Einsatz von Ajax wird auf dem Server oftmals Java benutzt, um für den Client optimiertes JavaScript zu erzeugen.

  Servlet Java Server Pages

  Für den plattformunabhängigen Austausch von Dokumenten auf Basis von XML entwickeln Oracle, die Apache Group und andere Firmen wie IBM Werkzeuge und Java-APIs. Entsprechende Projekte finden sich unter:

  XML: Extensible Markup Language

  `http://www.w3.org/XML/`[4] `http://xml.apache.org/`[5]

- Zur Steuerung vieler elektronischer Helfer (PDAs, Mobiltelefone, intelligenten Chipkarten) wird die Java Micro-Edition (Java ME) zur sicheren Programmierung dieser Geräte verwendet. Die speziellen Eigenschaften von Java verhindern dabei den Missbrauch des Geräts durch ein von Dritten geliefertes Programm (z. B. Missbrauch eines Handys als Abhöreinrichtung).

- Die Plugin-Technik ist heute in der Softwareentwicklung weit verbreitet. Dabei wird ein Programm in einen festen Teil (Kern) und Zusatzprogramme (Plugins) aufgeteilt, die den Kern für bestimmte Aufgaben anpassen. Java eignet sich hierfür besser als andere Programmiersprachen, weil das Kernprogramm die vom Plugin ausgeführten Operationen abfangen kann, bevor diese das Gesamtsystem destabilisieren können. Weiter fängt die JVM viele unzulässige Operationen des Plugins ab. Ein gutes Beispiel für

[2] Ajax: Asynchronous JavaScript and XML - Hilfsmittel zur Erstellung interaktiver browserbasierter Anwendungen.

[3] JDBC: Java Database Connectivity–Der Datenbank-Zugriffstandard in Java.

ein solches System ist das SAP-System, bei dem seit einigen Jahren neben ABAP auch Java als Erweiterungssprache unterstützt wird.

Auch Entwicklungsumgebungen wie NetBeans oder Eclipse benutzen eine solche Plugin-Architektur, bei welcher der Kern bestimmte zentrale Dienste anbietet und die Plugins den jeweils für eine Programmiersprache spezifischen Code enthalten. Diese Entwicklung geht soweit, dass selbst die elementaren Funktionen wie Editor oder Projektverwaltung inzwischen über Plugins geladen werden. So lässt sich mit NetBeans auch eine Warenwirtschaft oder ein Programm zur Unterstützung von Fluglotsen erstellen, beides Aufgaben, die kaum Ähnlickeit mit einer Entwicklungsumgebung haben.

- In der Ausbildung wird Java als konzeptionell einfache und vollständig objektorientierte Programmiersprache mit einer umfangreichen Bibliothek an Programmierschnittstellen geschätzt. Java wird zunehmend als primäre Programmiersprache in der Ausbildung an Schulen, Hochschulen und Universitäten eingesetzt.
- Der Zugriff auf Hardware und Systemkomponenten ist aus Java wegen der Plattformunabhängkeit und Architekturneutralität nicht möglich. Falls Sie in Ihrem Programm z. B. direkt den Prozessor einer Grafikkarte ansprechen wollen, bietet Ihnen Java hierfür keine unmittelbaren Möglichkeiten. Hier muss C oder im Zweifelsfall sogar Assembler eingesetzt werden. Java bietet für solche Zwecke z wei Schnittstellen an:
  - Mit dem Java Native Interface JNI können Sie aus Java-Programmen heraus auf speziell darauf abgestimmte C- oder C++-Funktionen zugreifen. Dabei ist für jede unterstützte Plattform eine separate C- oder C++-Bibliothek erforderlich, während die Java-Schnittstelle in der Regel gleich bleibt. JNI: Java Native Interface
  - Mit dem Java Native API JNA können Sie aus Java-Programmen auf die system-eigenen Funktionen zugreifen. In diesem Fall beschreiben Sie der JNA-Bibliothek, welche Parameter die Systemfunktionen erwarten und die Bibltiohek nimmt die Umsetzung vor. Die Java-Schnittstelle muss dann für jedes System angepasst werden. JNA: Java Native API
- Moderne Programme bestehen aus vielen Modulen. Jene Module, die auf Geschwindigkeit optimiert sind, sind in C oder C++ erstellt und werden für jede neue Plattform separat übersetzt, bei Bedarf auch angepasst. Andere Module, die z. B. für die Administration zuständig sind, und bei denen die Rechengeschwindigkeit nicht im Mittelpunkt steht, werden dagegen in Java programmiert.

2.5 Entwicklung von Java

Wie bereits erwähnt, begann die Entwicklung von Java Anfang der 90er Jahre. Die erste Version wurde im Jahre 1995 mit Java 1.0 veröffentlicht, die aktuelle Version trägt den Namen Java 6. Tabelle 2.1 enthält eine Liste Java-Versionen mit den Namen der Sprache und der Zahl an Paketen und Klassen. Letzteres zeigt deutlich die dynamische Entwicklung der Sprache:

Datum	Sprach-Version	Compiler-Version	Pakete & Klassen		Bemerkungen
Januar 1996	Java 1	JDK 1.0	8	212	Erste Version, im Netscape-Browser enthalten
Februar 1997	Java 1	JDK 1.1	23	503	Anonyme Klassen, Neue Ereignisbehandlung, JDBC, RMI
Dezember 1998	Java 2	J2SE 1.2	59	1520	Umbenennung des JDK zu SDK, Swing, Container
Mai 2000	Java 2	J2SE 1.3	76	1842	HotSpot JVM, CORBA-RMI, Debugger (JPDA)
Februar 2002	Java 2	J2SE 1.4	135	2991	Assert, Reguläre Ausdrücke, IPv6, NIO (New Input/Output), JPEG, WebStart, XML, Logging
September 2004	Java 5	J2SE 5.0	165	3278	Generische Typen, Annotationen, Enumerationen
Dezember 2006	Java 6	J2SE 6	202	3688	Skript-Support, Compiler-API, Spring Layout
Juli 2011	Java 7	J2SE 7	209	4005	JVM-Unterstützung für Python, Ruby
Mitte 2012	Java 8	J2SE 8			Module, Closures

Tabelle 2.1: Tabelle der bisherigen Java-Versionen

2.6 Java, JavaScript und JavaFX

Um es gleich am Anfang klar zu sagen: Java hat mit JavaScript nichts zu tun, es hat von Java nur die vier Buchstaben im Namen und einige wenige Bezeichnungen in der Syntax übernommen.

JavaScript wurde von Netscape als Sprache zur Erweiterung von HTML entwickelt. Der Anwendungsbereich von JavaScript ist auf Funktionen in HTML-Seiten beschränkt, wobei aber durch eine geschickte Kombination von JavaScript und einem XML-fähigen Browser (Ajax) die Funktionalität normaler Applikationen erreicht wird. JavaScript-Anweisungen werden immer innerhalb einer HTML-Seite vom Browser ausgeführt. Es wird kein Byte- oder Binärcode von JavaScript-Anweisungen erstellt.

JavaScript ist darüberhinaus stark browserabhängig und wegen der nicht vorhandenen Syntaxprüfung durch einen Compiler auch recht fehlerträchtig. Ein Fehler im Code kann jahrelang unentdeckt bleiben, bis irgendwann der fehlerhafte Code ausgeführt wird. Um dieses Problem zu umgehen, setzt z. B. Google in seinen Web-Seiten eine spezielle Technik ein, bei der ein Java-Programm JavaScript produziert. Wenn das Java-Programm korrekt übersetzt werden kann, ist auch das JavaScript-Programm korrekt, darüberhinaus kapselt der Übersetzungsprozess die Unterschiede der JavaScript-Dialekte.

JavaFX ist eine Sprache zur Darstellung von Animationen im Web und be-

findet sich aktuell noch im Stadium der Erprobung, ähnlich wie Adobe Flash, Microsoft Silverlight, etc.

2.7 Java und OpenSource

Java war lange Zeit weder proprietär noch offen. Die Technologie konnte von anderen Firmen (u.a. von IBM, Oracle (damals noch nicht Eigentümer der Sprache), Microsoft) lizensiert werden, aber es war nicht erlaubt, die Sprache zu erweitern. Auch wurden die Quelltexte des Compilers, der JVM und von Teilen der Laufzeitbibliothek nicht veröffentlicht. Dies resultierte nicht zuletzt aus dem Bemühen von Microsoft, die Sprache nach der Integration in den Internet Explorer um Windows-spezifische Sprachelemente zu erweitern. Hätte Microsoft mit dieser Strategie Erfolg gehabt, hätte der Grundsatz "Write once, run anywhere" für viele Programme nicht mehr gegolten, da Programme, die diese Elemente genutzt hätten, nur noch in der Microsoft-JVM lauffähig gewesen wären.

Sun hat in Folge darum gekämpft, Microsoft diese Modifikation von Java zu verbieten und war zunächst nicht geneigt, Java unter eine OpenSource-Lizenz zu stellen. Erst im November 2006 wurde die gesamte Java-Technologie (mit Ausnahme weniger Teile) unter die GPL gestellt. Dabei dürfen Programme, die unter Benutzung von Java bzw. seiner Klassenbibliotheken erstellt wurden, unter jeder beliebigen Lizenz publiziert werden.

GPL: GNU General Public License

Mit dieser Massnahme wurde Java zwar OpenSource, es wurde jedoch mit dem Java Community Process (jcp.org) ein Mittel gefunden, künftig einseitige Änderungen der Sprache zu unterbinden. Mittels eines JSR (Java Specification Request) kann theoretisch jeder Nutzer von Java Änderungen an der Sprache vorschlagen. Allerdings darf man die so geänderte Sprache nicht unter dem Namen Java veröffentlichen, wenn der JSR nicht vom JCP angenommen wurde. So bleibt der Satz "Write once - run anywhere" trotzdem gültig.

2.8 Bewertung und Fazit

Java hat heute eine große Bedeutung in der Software-Entwicklung und Programmierung. Der Erfolg von Java ist dabei sicherlich nicht nur auf innovative Ideen zurückzuführen.

Viele Konzepte, die von Sun Microsystems/Oracle mit Java erfolgreich vermarktet wurden und werden, sind bereits seit vielen Jahren in der Software-Entwicklung bekannt.

Der Erfolg von Java beruht im großen Maße auch auf dem Internet-Boom. Java wurde als Programmiersprache für das Internet bekannt, das Internet wurde als Vermarktungs- und Infrastrukturbasis genutzt, und für alle aktuellen und "heißen" Themen im Internet wurden Java-Lösungen präsentiert. Dass dabei nicht alle in der Dynamik der Entwicklung präsentierten Lösungen ausgereift waren, und nicht jedes Konzept die gewünschte Verbreitung gefunden hat, hat dem Gesamterfolg von Java keinen Abbruch getan.

Java wird häufig auch nicht nur als plattformunabhängige, sondern auch als firmenunabhängige Programmiersprache angesehen, und Java-Anwendungen werden als Gegengewicht zur Marktbeherrschung der Firma Microsoft präsentiert. Die Plattformunabhängigkeit und das Fehlen firmen- und herstellerspezi-

fischer Merkmale gilt zwar uneingeschränkt für Java, zu bedenken ist jedoch, dass Java keine freie Software ist, sondern dass die primäre Entwicklung und die Lizenzrechte zentraler Bestandteile immer noch bei Sun bzw. beim Rechtsnachfolger Oracle liegen.

Im Vorfeld der Übernahme war viel spekuliert worden, ob Oracle seine Rechte an Java (die Sprache ist zwar vollständig OpenSource, die konkrete Implementierung aber nicht) deutlich aggressiver nutzen würde, als Sun dies bis anhin getan hat. Hier wird die Zukunft zeigen, ob diese Spekulationen sich bewahrheiten.

Trotz dieser, vielleicht auch kleinerer Bedenken: Ein Ende des Trends in Richtung Java ist zur Zeit nicht absehbar.Renommierte Hersteller und Organisationen beteiligen sich zunehmend mit eigenen Produkten und Entwicklungen an der Verbreitung von Java. Viele große Software-Firmen haben Projekte mit Java gestartet und insbesondere im kommerziellen Bereich (Electronic Commerce, B2B, Online Banking, Networking und Kommunikation) werden Applikationen erfolgreich mit Java realisiert.

Dem Programmier-Anfänger bietet Java die Möglichkeit, moderne Konzepte der Programmierung und Software-Technologie kennen zu lernen.Wer heute im Bereich aktueller Software-Entwicklung kompetent mitwirken will, wird Java kaum vernachlässigen können.