

Computerkollaboration (CCL)

Computer

Erstellung

Erstellung

Erstellung von Computer

Erstellung von Produkten

Erstellung von Produkten

Erstellung von Produkten

Erstellung von Produkten

ITWissen  
Das große Online-Lexikon  
für Informationstechnologie

**JAVA**  
**A. NIEMANN (Hrsg.)**

## Inhalt

---

**AspectJ****EJB**, *enterprise JavaBeans***J2EE**, *Java 2 enterprise edition***J2ME**, *Java 2 micro edition***J2SE**, *Java 2 standard edition***JAAS**, *Java authentication and authorization service***JACL**, *Java command language***Java****Java 2D API****Java 3D API****Java Card****Java Communications API****Java Package****Java Plug-in****Java web start****JavaBeans****Javadoc****JavaScript****JCE**, *Java cryptography extension***JCP**, *Java community process***JDBC**, *Java database connectivity***JDK**, *Java development kit***JFC**, *Java foundation class***JHTML**, *Java HTML***JMS**, *Java message service***JMX**, *Java management extension***JNDI**, *Java naming and directory interface***JPDA**, *Java platform debugger architecture***jQuery****JRE**, *Java runtime environment***JRuby****JSP**, *Java server pages***JSR**, *Java specification request***JVM**, *Java virtual machine***Swing**

Impressum:

Herausg.: Alexander Niemann

Java

Copyright 2010

DATACOM-Buchverlag GmbH

84378 Dietersburg

Alle Rechte vorbehalten.

Keine Haftung für die angegebenen Informationen. Das E-Book ist urheberrechtlich geschützt und darf nicht auf fremden Websites ins Internet oder in Intranets gestellt werden.

Produziert von Media-Schmid

[www.media-schmid.de](http://www.media-schmid.de)

**AspectJ** Bei AspectJ handelt es sich um eine Erweiterung der Programmiersprache *Java* um spezielle Konstrukte zur aspektorientierten Programmierung (AOP). AspectJ wurde bereits 1996 von einem Team am Xerox Palo Alto Research Center - maßgeblich beeinflusst von Georg Kiczales - entwickelt. AspectJ ist seit dem Jahr 2002 auch Teil des Eclipse Projekts, welches aktuelle Development Tools zur Verfügung stellt. Mit Einführung der AOP wird Java nur um eine bestimmte Klasse Aspect ergänzt, die die AOP-spezifischen Methoden kapselt. Der funktionale Java-Programm-Code wird an den zuvor definierten Verwebungspunkten mit einem Aspekt verwoben. Für die Ausführung von AspectJ-Programm-Code werden neben dem *Java Runtime Environment* (JRE) ein entsprechender Compiler - auch Weaver genannt - und die AspectJ Runtime Library benötigt. Ein Aspect ist dadurch beim Start eines Programms bereits gegenwärtig, da er durch den Weaver in den vorhandenen Code eingewebt wurde, und somit physischer Teil der Anwendung ist und braucht daher auch nicht separat gestartet zu werden.

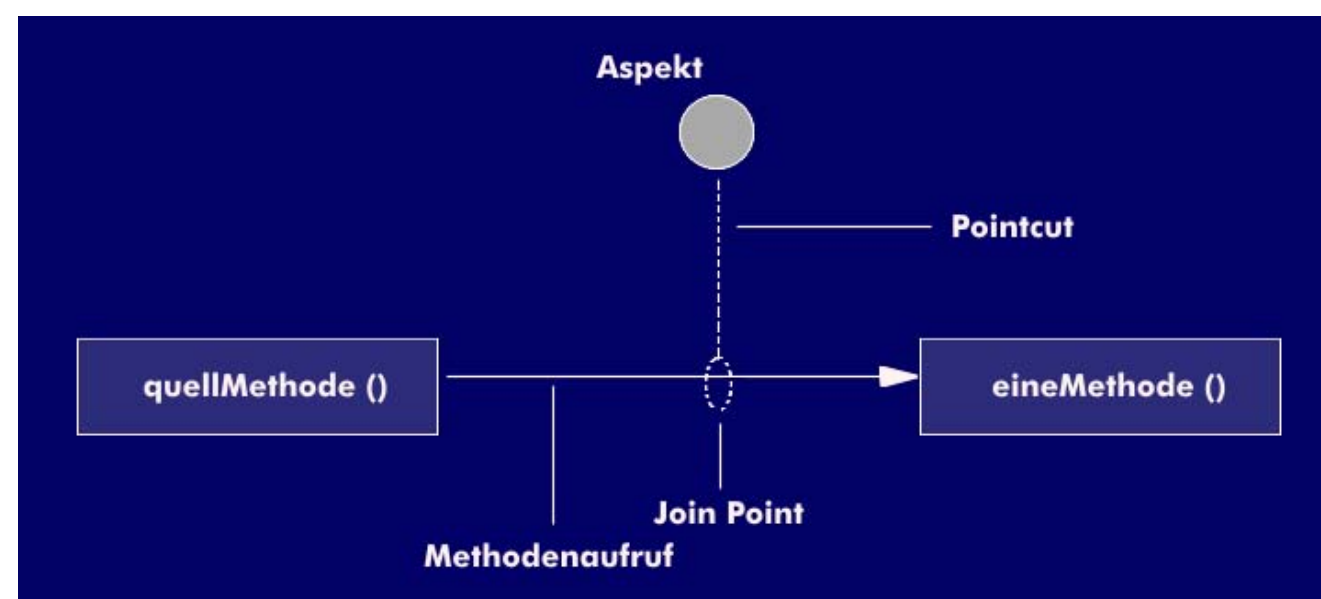
AspectJ war die erste aspektorientierte Programmiersprache, und da sie zudem noch auf der weit verbreiteten Programmiersprache Java basiert, besitzt auch AspectJ ebenso eine weite Verbreitung sowie eine hohe Akzeptanz.

Für eine verständliche Erläuterung von AspectJ, müssen zunächst die folgenden Begriffe erläutert werden. Aspekt. Aspekte implementieren sogenannte Crosscutting Concerns - auch Querschnittsfunktionalitäten genannt. Damit ist eine generelle Funktion wie Logging, Authentifizierung etc. gemeint.

Join Point. Mit Join Point wird die Stelle innerhalb einer Software bezeichnet, an denen sich der AspectJ-Code in den Programmfluss einklinken soll. Join Points können Ausführungen von Methoden sein, die Ausführung sowie der Aufruf eines Konstruktors oder die Referenz auf Datenstrukturen (Variablen).

Point Cut. Als Point Cut wird eine Menge von Join Points - auch Verwebungspunkte genannt- bezeichnet. Diese Menge kann leer sein oder einen, mehrere oder alle Join Points enthalten.

Advice. Dies ist ein spezifischer Begriff von AspectJ und definiert, was zu welchem Zeitpunkt an den Join Points eines Point Cuts umgesetzt wird. Advices bestehen aus ganz normalem Java Programmcode. Für den Zeitpunkt des Einfügens des AspectJ-Codes können festgelegt werden: before, after und around.



Inter Type Declaration. Mit Anwendung der AOP ist eine dynamische Erweiterung von Klassen möglich, was als Inter Type Declaration bezeichnet wird. Die Abbildung verdeutlicht den Zusammenhang zwischen Aspekten, Point Cuts und Join Points. Dort ist der Join Point eine im Programmablauf fest definierte Stelle, die den Aspekt- und den nicht-

Aspekte, Point Cuts und Join Point

aspektorientierten Code koordiniert.

Die aspektorientierte Programmierung erweitert die vorhandenen Java-Klassen um eine einzige spezielle Klasse Aspect, die dann ihre eigenen Datenstrukturen und Methoden kapselt. Ein zentrales Element dabei sind die Point Cuts, die u.a. aus den o.g. Methoden aufgebaut sein bzw. noch durch eine Reihe weiterer Auswahlmethoden gebildet werden können. Von AspectJ wird zur Laufzeit in die Struktur der Klassen einer Anwendung eingegriffen, ohne dass die Anwendung speziell vom Vorhandensein der Aspekt-Klasse Kenntnis hat. Da AspectJ u.a. die Verwendung von Wildcards hinsichtlich der Auswahl der Klassen, Methoden u.a. durch Pattern-Matching gestattet, muss auch der Aspekt von der Klasse nichts wissen, auf welche er letztendlich zugreift. Die Umsetzung eines Aspektes - das Einweben eines Aspektes in den Java-Programmcode - erfolgt dann entweder mit der Compilierung oder zur Laufzeit, was dann Auswirkungen auf die Performance einer Anwendung hat.

AspectJ unterstützt generische und parametrisierte Typen in pointcuts, intertype declarations sowie bei der Vererbung. Seit der Version 5 von AspectJ können Features wie Annotations, Generics, Autoboxing und Unboxing verwendet werden. Annotationen können Aspekte und innerhalb von Aspekten Methoden, Felder, Konstruktoren, Advices etc. mit Metadaten kennzeichnen. Annotationen können auch in Pointcuts als zusätzliche Bedingungen für Verwebungspunkte angegeben werden.

Nachfolgend ein Beispiel zur Darstellung der Notation von AspectJ. Dabei definiert der Aspekt Trace einen pointcut für den Aufruf aller Methoden mit add mit beliebigen Argumenten sowie beliebigem Rückgabewert. Zudem werden von Trace zwei advices definiert.

Für die Arbeit mit AspectJ sind grundsätzliche folgende Komponenten in einem Paket zusammengefasst:

- Der AspectJ Compiler (Weaver) ajc.
- Der AspectJ Interpreter aj.
- Die AspectJ Runtime Library ASPECTJRT\_LIB.

Die Verwebung einer Java-Klasse mit einem Aspekt erfolgt statisch auf Basis des Quellcodes mit Hilfe des Compilers ajc. Dazu werden von ajc zunächst die Java-Klassen und die Aspekte compiliert und diese dann auf Basis des so generierten Bytecodes verwoben. Um dann die so entstandene Anwendung zu nutzen, genügt dann das bekannte Java Runtime Environment (JRE).

Zu den Vorteilen von AspectJ zählen eine bessere Modularisierung der Anwendung, die Möglichkeit der Wiederverwendung von Code und es kann sehr flexibel genutzt werden, da schnell und ohne Verluste von Performance zwischen Java- und AspectJ-Code umzuschalten ist.

Denen stehen die folgenden Nachteile gegenüber: Die Definition von Point Cuts setzt eine uneingeschränkte Kenntnis der Anwendungsklassen voraus, AspectJ-Code ist nicht automatisch restrukturierbar, die Erstellung eines Point Cuts mit zugehörigem Advice ist häufig aufwendiger als direkter Java-Code, die Nutzung von AspectJ setzt bei komplexeren Projekten eine spezielle Entwicklungsumgebung voraus und die Syntax von AspectJ weicht teilweise erheblich von der Java-Syntax ab.

AspectJ unterliegt einer ständigen weiteren Entwicklung und wird von einer Reihe von Entwicklungsumgebungen wie Emacs, JBuilder und Netbeans unterstützt. Außerdem natürlich von Eclipse, bei dem sowohl eine AspectJ-Projekt und ein separates AspectJ Development Tools-Projekt läuft. Letzteres entwickelt für AspectJ Eclipse Plugins. Durch die Integration als Plugin in die Entwicklungsumgebung Eclipse

```
public aspect Trace {
    pointcut add () : call (public * add (..));
    before () : add () { ..... }
    after () : add () { ... }
}
```

*Darstellung der Notationen von AspectJ*

ist ein spezieller Editor für AspectJ verfügbar sowie eine Reihe weiterer Funktionen:

- Automatische Generierung von AspectJ-Dokumentation.
- Die Verwebungspunkte werden im Quellcode angezeigt.
- Durch einen Wizard gesteuert, können Aspekte konfiguriert werden.
- Build-Dateien können bearbeitet und verwaltet werden.
- Das Debugging kann auf grafischem Wege erfolgen.
- Der Quellcode wird automatisch vervollständigt.

Eine weitere Implementierung von AspectJ ist der AspectBench Compiler (auch abc genannt), der sowohl ein Referenz-Compiler zu ajc ist, als auch als ein ergänzendes Framework zur Implementierung von Erweiterungen zu AspectJ. Diesbezüglich hat abc auch die Optimierung des Sprachkerns von AspectJ zum Ziel. Dabei handelt es sich bei abc um ein universitäres Projekt aus den USA, Kanada und Dänemark. Dazu baut abc auf den beiden eigenständigen Frameworks Polyglot und Soot auf. Zudem wurde die komplette Laufzeit-Bibliothek aus ajc entnommen. Ein wesentlicher Unterschied ist, dass das Compilieren von Aspekten nicht separat unterstützt wird.

<http://www.eclipse.org/aspectj/>

<http://www.aspectbench.org>

## **EJB, enterprise JavaBeans**

Enterprise *JavaBeans* (EJB) ist ein clientseitiges Komponentenmodell für die Programmiersprache *Java*, das den Clients die Funktionen und Eigenschaften zur Verfügung stellt. Es ist komplexer als die *JavaBeans* und umfasst die Prinzipien in denen *Java*-Anwendungen auf dem Server arbeiten. Die verteilten *JavaBeans* sichern die Transaktionen und sind skalierbar.

## **J2EE, Java 2 enterprise edition**

Die *Java 2 Enterprise Edition* (J2EE), eine Erweiterung der *Java 2 Standard Edition* (J2SE), ist eine *Java*-Plattform für Unternehmensanwendungen. Der J2EE-Standard arbeitet mit Basiskomponenten, mit denen die Anwendungen realisiert werden. Dazu gehört der *Java Message Service* (JMS), der die Programmierschnittstellen für die Kommunikation zwischen den verteilten Komponenten bereitstellt. Ist eine Anwendung J2EE-konform, kann sie auf andere J2EE-Application-Server portiert werden.

Da J2EE-Umgebungen Webservices unterstützen, bietet sich diese Plattform für die Enterprise Application Integration (EAI) an.

J2EE wird häufig für umfangreiche Anwendungen im Enterprise-Bereich eingesetzt.

## **J2ME, Java 2 micro edition**

Die *Java 2 Micro Edition* (J2ME) ist eine verschlankte Version von *J2EE* und *J2SE*. J2ME ist optimiert für Mobilgeräte wie Handys, Handhelds, Palmtops, PDAs, usw. und für Kommunikationsendgeräte wie Settop-Boxen oder auch Telefonen. J2ME besteht aus den Modulen Konfiguration und Profile und verfügt über einen geringen Speicherbedarf, einen reduzierten Befehlssatz und kurze Programmlaufzeiten.

## **J2SE, Java 2 standard edition**

Die *Java 2 Platform, Standard Edition* (J2SE) ist der Rahmen für die Entwicklung von *Java*-Programmen für Desktop- und Server-Systeme. Gleichzeitig dient sie als Grundlage für die *Java 2 Platform, Enterprise Edition* (*J2EE*). Sie ist in zwei Hauptbereiche unterteilt:

- Core Java Technology ist die Grundlage aller Java-Technologien und enthält neben dem Sprachkern grundlegende Funktionen und APIs wie beispielsweise Sicherheitsmechanismen (*JAAS* und *JCE*), Datenbankzugriffe (*JDBC*), Debugging (*JPDA*), Dokumentation (*javadoc*), Internationalisierung, Verzeichnisdienste (*JNDI*) und entfernte Prozeduraufrufe (*RMI*).
- Desktop Java Technology wird zur Entwicklung von Desktop-Anwendungen mit grafischen Benutzeroberflächen benötigt. Dazu gehören vor allem die *Java Foundation Classes* (*JFC*), das Komponentenmodell *JavaBeans* und die Unterstützung der Anwendungsverteilung (Deployment) durch *Java Web-Start* und *Java Plug-in*.

Eine besondere Herausforderung für eine plattformunabhängige Sprache ist die Darstellung von grafischen Benutzeroberflächen (GUI), die sehr plattformspezifisch sind. J2SE bringt dazu die APIs *AWT* und *Swing* mit, deren Benutzung allerdings nur erfahrenen Softwareentwicklern gelingt. Denn beide APIs folgen sehr streng dem mächtigen aber komplizierten Model-View-Controller (MVC) Entwurfsmuster. Von Drittanbietern verfügbare Bibliotheken wie beispielsweise *SWT* schlagen Mittelwege ein und vereinfachen die GUI-Entwicklung mit Java.

J2SE kann auch für die Entwicklung von Embedded-Software und Echtzeitanwendungen verwendet werden. Für letzteres bietet Sun das *Java Real-Time System* (*JRTS*) an.

Des Weiteren gibt es eine Vielzahl von optionalen APIs, die zusammen mit der J2SE verwendet werden können und den Einsatzbereich erweitern. Dazu zählen beispielsweise die *Java Management Extensions* (*JMX*), *JMX Remote API*, *Java Communications API*, *Java Telephony API*, *Java Media Framework* (*JMF*), *Java 3D API* und *Java Advanced Imaging API* (*JAI*).

<http://java.sun.com/j2se/index.jsp>

## **JAAS, Java authentication and authorization service**

Der *Java Authentication and Authorization Service* (*JAAS*) ist ein Satz von Java-APIs, mit deren Hilfe ein Anwendungsprogramm Benutzer authentifizieren und den Zugriff auf Programmteile kontrollieren kann. Die Core Java Technology von *Java 2 Standard Edition* (*J2SE*) enthält neben dem Sprachkern auch die APIs für die Sicherheitsmechanismen (*JAAS*).

## **JACL, Java command language**

Die *Java Command Language* (*JACL*) ist ein Interpreter für die Scriptsprache *Tool Command Language* (*Tcl*), die vollständig in Java implementiert ist. Dabei ist *Tcl* als Open-Source verfügbar und aufgrund ihres einfachen Aufbaus, leicht zu erlernen ist. Die Java-Implementierung von der *Tcl* integriert jedoch nicht das zugehörige "Toolkit" für die Programmierung von grafischen Benutzeroberflächen (GUI). *JACL* steht in der Version 1.4.1 aus dem Jahr 2008 zur Verfügung.

*JACL* wird verwendet um Scripting-Funktionalität in eine vorhandene Java-Anwendung zu integrieren. Damit ist *JACL* ideal für Anwender, die sich nicht in komplexe Strukturen eines Java-Programms einarbeiten wollen. *JACL* definiert jedoch auch beiderseits ein *Application Programming Interface* (*API*) mit dem von *Tcl* auf Java und umgekehrt zugegriffen werden kann. *JACL* enthält umfangreiche Funktionen, die die Kommunikation zwischen einem Java-Interpreter und einem *Tcl*-Interpreter vereinfachen.

Als Vorteile von *JACL* können genannt werden:

- Komfortable Schnittstelle für Java-Scriptsprachen,

- Möglichkeit der objektorientierten Erweiterung durch Zugriff auf Java Objekte und Bibliotheken. Denen stehen die folgenden Nachteile gegenüber:
  - JACLS funktionieren nicht im Zusammenhang mit Applets,
  - Aufgrund der fehlenden Objektorientierten Eignung von JACL gibt es eine Möglichkeit Klassen zu erweitern,
  - Die Verwendung von JACL hat eine geringe Performance zur Folge.

Das Tcl/Java-Projekt hat das Ziel der Integration von Tcl und der Java-Plattform. Dazu gibt es mit TclBlend eine weitere Möglichkeit der Kopplung von Java und Tcl. Dabei werden der in der Programmiersprache C implementierte Tcl-Interpreter und die *Java Virtual Machine* (JVM) über das *Java Native Interface* (JNI) miteinander verbunden. Da dadurch aber ein höherer Aufwand hinsichtlich der Compilierung entsteht, ist JACL die einfachere Alternative.

<http://tcljava.sourceforge.net>

**Java** Java ist eine objektorientierte und plattformunabhängige Programmiersprache, deren Grundlagen in der ersten Hälfte der 1990er Jahre von Sun Microsystems entwickelt worden sind. Die Syntax von Java lehnt sich stark an C++ an, obwohl die Sprache selbst sich in vielen Punkten unterscheidet.

Bis zur Einführung von Java 1995 wurden im akademischen Betrieb hauptsächlich Sprachen wie Eiffel und Smalltalk verwendet, um objektorientierte Programmierung zu lehren. Beide sind durch Java stark zurückgedrängt worden. Im kommerziellen Bereich dominierte C++ die objektorientierte Entwicklung, das aber ebenfalls in hohem Maße durch Java ersetzt worden ist.

Java nimmt in der klassischen Unterscheidung zwischen interpretierten und kompilierten Programmiersprachen eine Sonderstellung ein. Der Quelltext wird zwar kompiliert, aber nicht für ein spezielles Zielsystem. Stattdessen entsteht ein Zwischencode (Bytecode), der von einer *Java virtual Machine* (JVM) ausgeführt wird. Deshalb muss auf jedem Zielsystem neben dem eigentlichen Java-Programm auch die Laufzeitumgebung *JRE* (Java Runtime Environment) installiert werden.

Neben den für Programmiersprachen üblichen Sprachelementen (reservierte Wörter, Variablen, elementare Datentypen und Kontrollstrukturen) unterstützt Java nahezu alle Techniken der Objektorientierung:

Geheimnisprinzip, Botschaften, Vererbung, Polymorphie, und Überladung. Explizit nicht unterstützt wird

Mehrfachvererbung, die als übermäßig komplex und in der Praxis unbedeutend angesehen wird. Stattdessen wird auf das Entwurfsmuster der Interfaces verwiesen. Java deckt viele Themenbereiche ab und wird durch eine Vielzahl von Bibliotheken (APIs)



Java mit seinen Plattformen

ergänzt. Deshalb benennt "Java" heute oft nicht nur die Sprache selbst, sondern dient als Oberbegriff für alle zugrunde liegenden Technologien. Diese sind durch die Plattformunabhängigkeit und das breite Verwendungsspektrum der Sprache geprägt. Sun unterscheidet dabei im Wesentlichen folgende Themenbereiche:

- Core/Desktop umfasst den Sprachkern und die wichtigsten Technologien rund um Anwendungsprogramme für PC. Die Bezeichnung für diesen Bereich ist *Java 2 Platform, Standard Edition (J2SE)*.
- Enterprise/Server umfasst Technologien zur Entwicklung von mehrschichtigen Softwaresystemen und Webservices. Die Bezeichnung für diesen Bereich ist *Java 2 Platform, Enterprise Edition (J2EE)*.
- Mobile/Wireless behandelt Technologien rund um Klein- und Kleinstgeräte wie Mobiltelefone und Handheld-Computer. Die Bezeichnung für diesen Bereich ist *Java 2 Platform, Micro Edition (J2ME)*.
- Java Card zielt auf Smartcards, auf denen Java-Programme ausgeführt werden können.

Entgegen der landläufigen Meinung Java sei eine spezielle Sprache für Internetanwendungen, kann diese Technologie also nahezu in allen Softwarebereichen eingesetzt werden. Eine besondere Eignung für verteilte Softwaresysteme ist jedoch tatsächlich vorhanden.

Die Weiterentwicklung von Java wird durch das Standardisierungsgremium *Java Community Process (JCP)* gesteuert. <http://www.sun.com/>, <http://java.sun.com/>

**Java 2D API** Die *Java 2D API* erlaubt die Verarbeitung von zweidimensionalen Grafiken und Bilder in Java-Programmen. Es handelt sich um einen Satz von Klassen für die Bearbeitung von Linienarten, Text und Bilder in einem umfassenden Modell. Mit Java 2D können vordefinierte Objekte erstellt und für die Grafikerstellung genutzt werden.

Die Programmierschnittstelle (API) unterstützt die Bildbearbeitung und den Alphakanal, ebenso wie die Definition des Farbraums und der Bildschirmdarstellung. Java 2D API hat viele Elemente für die grafische Bildbearbeitung. Dazu gehören geometrische Formen, das Zerlegen von Farben, Rendering, das Füllen von Flächen mit Farben und Texturen, die Formatierung von Text und das Clipping.

Für dreidimensionale Grafikverarbeitung gibt es die *Java 3D API*.

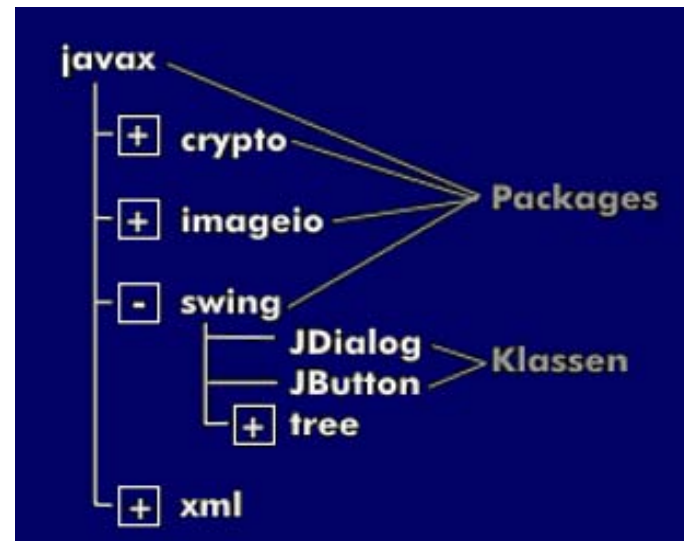
**Java 3D API** Die *Java 3D API* erlaubt die Verarbeitung von dreidimensionalen Grafiken in Java-Programmen. Für zweidimensionale Grafikverarbeitung gibt es die *Java 2D API* und die *Java Advanced Imaging API (JAI)*.

**Java Card** Die *Java-Card-Technologie* ermöglicht es, Java-Programme auf Smartcards zu installieren und auszuführen. Auch andere Geräte, die mit vergleichsweise wenig Speicher und CPU-Leistung auskommen müssen, gehören zur Zielgruppe von Java Card.

<http://java.sun.com/products/javacard/index.jsp>

**Java Communications API** Die *Java Communications API* ermöglicht die Kommunikation mit Hardware-Schnittstellen aus Java-Programmen heraus. So können beispielsweise parallele und serielle Schnittstellen und USB-Schnittstellen verwendet werden.

## Java Package



Java-Packages, Auszug aus J2SE

Java-Programme und APIs bestehen aus Ansammlungen von vielen einzelnen Dateien, die innerhalb der objektorientierten Programmierung jeweils eine Klasse repräsentieren. Um dieser Klassensammlung eine Struktur geben zu können und so die Les- und Benutzbarkeit für Menschen zu verbessern, wurden Java Packages eingeführt.

Dabei handelt es sich um eine hierarchische Struktur, in der die Klassen (= Dateien) abgelegt werden. Jedes Package hat eine oberste Ebene, in der beliebig viele Klassen und weitere Ebenen liegen können. Für jede untergeordnete Ebene gilt wiederum das gleiche. So entstehen Pakete, die Baumstrukturen abbilden und Klassen sinnvoll adressierbar machen.

**Java Plug-in** Die Java Plug-in Technologie ist Bestandteil der *JRE* und ermöglicht die Verbindung einer *JVM* mit einem Browser. Dieser bekommt dadurch die Möglichkeit, aus dem Internet geladene Applets auszuführen. Diese Funktionalität gehört zum Bereich Softwareverteilung (Deployment). Um andere Java-Programme als Applets über das Internet zu verteilen, kann *Java Web Start* verwendet werden.

**Java web start** Die Java Web Start-Technologie ist Bestandteil der *Java Laufzeitumgebung* (JRE) und ermöglicht die Verteilung von Java-Programmen über Netzwerke wie das Internet. Dabei wird die komplette Anwendung heruntergeladen und lokal ausgeführt, ohne dass ein Benutzer manuell eingreifen muss. Bei jedem Programmstart wird automatisch geprüft, ob eine aktuellere Version zur Verfügung steht und geladen werden muss.

Diese Funktionalität gehört zum Bereich Softwareverteilung (Deployment). Um nicht allein stehende Java-Programme (Applets) über das Internet zu verteilen, kann *Java Plug-in* verwendet werden.

**JavaBeans** JavaBeans sind in Java programmierte Softwarekomponenten, die speziell für Wiederverwendbarkeit entworfen werden. Eine JavaBean stellt einen Satz von Funktionen und Eigenschaften zur Verfügung und kommuniziert diese auch nach außen. Die kommunikative Basis ist die Remote Method Invocation (RMI) und das IIOP-Protokoll von Corba. So ist es beispielsweise möglich, zur Laufzeit einer Anwendung eine JavaBean zu laden und zu benutzen, über deren Funktionen und Eigenschaften man zur Entwicklungszeit noch nichts wusste. So entstehen dynamische Softwaresysteme aus vorgefertigten Komponenten, auf deren Funktionsweise man sich verlassen kann und die nicht selbst programmiert werden müssen.

<http://java.sun.com/products/javabeans/index.jsp>

**Javadoc** javadoc ist ein Programm, das bei der Entwicklung von Java-Programmen benutzt wird und zum *Java Development Kit* (JDK) gehört. Es ist in der Lage, aus speziell gekennzeichneten Kommentaren innerhalb des Quelltextes technische Dokumentationen im HTML-Format zu erstellen. Diese Dokumentationen enthalten Informationen über die Klassenhierarchie (Vererbung in objektorientierter Programmierung),

verfügbare Methoden und vor allem ergänzenden Text, den Programmierer hinzugefügt haben um die Funktionsweise zu erläutern.

**JavaScript** JavaScript ist eine vielseitige Programmiersprache, mit der sich komplexe Anwendungen entwickeln lassen. Interpreter für JavaScript sind in allen modernen Web-Browsern integriert, daher ist die Sprache erheblich verbreitet. Daneben wird JavaScript zum Scripten einiger populärer Software genutzt. JavaScript wird in der Hauptsache Client-seitig verwendet und es ist es damit möglich, auf Webseiten dynamisch Einfluss zu nehmen.

JavaScript wurde durch die European Computer Manufacturers Association (ECMA) unter der Normierung ECMA-262 (ECMAScript) standardisiert. Aus Gründen der Sicherheit für das individuelle System hat JavaScript keinerlei direkten Zugriff auf dessen Dateien. Zudem muss auf jedem System der JavaScript vor dessen Anwendung aktiviert sein. Neben dem rein funktionalen Stil unterstützt JavaScript die Methodik der objektorientierten Programmierung.

Zunächst ist JavaScript eine Scriptsprache, deren Programmcode durch die Funktionalität eines Browsers im ersten Schritt interpretiert und dann direkt ausgeführt wird. Dabei definiert die Bezeichnung Scriptsprache eine für eher überschaubare Software-Aufgaben gedachte Programmiersprache. JavaScript ist eine vielseitige und flexible Sprache mit objektorientierten Fähigkeiten.

Ursprünglich wurde die Sprache entwickelt, um die rein statisch ausgerichteten HTML-Seiten aus Sicht des Anwenders um Möglichkeiten zu erweitern, dynamisch auf Webseiten Einfluss nehmen zu können. Die Entwicklung des Internets sorgte dann für eine rasche Verbreitung und eine wichtige Rolle von JavaScript. Dabei gibt es inzwischen auch unterschiedliche andere Möglichkeiten, dynamische Webseiten zu gestalten. Für die Entwicklung von JavaScript war es dabei sicherlich von Vorteil, dass die Sprache unabhängig von einer spezifischen Plattform ist.

Für eine Sprache, deren eigentliches Anwendungsgebiet das Internet mit den unterschiedlichsten Systemen und Plattformen ist, ein enorm hoch zu bewertender Aspekt. Ein weiterer vorteilhafter Gesichtspunkt ist, dass es sich bei JavaScript um eine im Wesentlichen Client-seitig verwendete Programmiersprache handelt. Das bedeutet, dass die Scripte vom Web-Browser des Clients - unabhängig vom Webserver - ausgeführt werden. Um dies zu realisieren wird der allgemeine Sprach-Kernel um Objekte im Sinne einer programmierbaren Schnittstelle erweitert, die es dann ermöglichen, auf Ereignisse (Events) wie z.B. Benutzereingaben zu reagieren, den Browser in seiner Funktionalität zu steuern oder beispielsweise auf Daten des HTML-Dokuments zuzugreifen.

**JCE, Java cryptography extension** Die *Java Cryptography Extension (JCE)* ist ein Sammlung von *Java Packages*, die kryptografische Verfahren implementieren. Dazu gehören die Verschlüsselung von Daten sowie Schlüsselerzeugung und -verifizierung mit verschiedenen Verfahren wie beispielsweise dem RSA-Verfahren.

**JCP, Java community process** Alle Weiterentwicklungen an der Programmiersprache *Java* unterliegen der Kontrolle des *Java Community Process (JCP)*, einem unabhängigen Standardisierungskomitee. In diesem Komitee sind sowohl kommerzielle Unternehmen als auch nicht-kommerzielle Organisationen

vertreten. Durch die Zusammensetzung soll erreicht werden, dass Vorschläge zur Weiterentwicklung möglichst unabhängig und objektiv geprüft werden. Erfolgversprechende und augenscheinlich sinnvolle Änderungen werden in *Java Specification Requests* (JSR) beschrieben und weiterverfolgt. Dabei findet auch eine Veröffentlichung im Internet statt, um für Transparenz nach außen zu sorgen.  
<http://www.jcp.org>

## JDBC, Java database connectivity

*Java Database Connectivity* (JDBC) ist die Java-Anwendungsschnittstelle (API) für Datenbanken, vergleichbar *Open Database Connectivity* (ODBC). Über diese Schnittstelle können SQL-Befehle ausgeführt und mit SQL-basierten Datenbanken kommuniziert werden. Über die JDBC-Schnittstelle kann aus Java-Programmen heraus auf beliebige Datenbanken zugegriffen werden. JDBC gehört neben *Open Database Connectivity* (ODBC) und dem Framework *Java Data Objects* (JDO) zu den Datenbankstandards.

## JDK, Java development kit

Das *Java Development Kit* (JDK) der Firma Sun bündelt eine Reihe von kommandozeilenorientierten Tools, die zur Erstellung, Kompilierung und zur Ausführung von Programmen in der Programmiersprache Java verwendet werden. Die JDK enthält zudem umfangreiche Bibliotheken, Quellcodes sowie Beispielprogramme, die für die Entwicklung eigener Software eine hilfreiche Basis zur Verfügung stellen. In der für verschiedene Einsatzgebiete zur Verfügung stehenden JDK ist keine grafische Entwicklungsumgebung enthalten. Im Januar 2010 steht die JDK in der Version 6, Update 18 unter der GNU General Public License (GPL) zur Verfügung. Gleichfalls sind unter dem u.g. Link die ausführlichen API- und Tool-Dokumentationen verfügbar.

Die JDK beinhaltet keine eigene grafische Entwicklungsumgebung. Doch haben verschiedene Hersteller seit der Entstehung von Java grafische Entwicklungsumgebungen für Java entwickelt. Hier sei beispielsweise auf die komfortablen Entwicklungsumgebungen Eclipse und NetBeans (ebenfalls ein kostenloses SUN-Produkt) verwiesen.

In Abhängigkeit vom jeweiligen Verwendungszweck liegt die JDK für verschiedene Einsatzgebiete vor. Standard Edition (J2SE). Die Java 2 Standard Edition wird zur Entwicklung und Ausführung von Java-Programmen für Standard-Anwendungen eingesetzt.

Enterprise Edition (J2EE). Mit der Java 2 Enterprise Edition werden zusätzliche Bibliotheken für verteilte Anwendungen und Web-Services angeboten.

Micro Edition (Java ME). Die Java Micro Edition ist für Anwendungen auf kompakten, tragbaren Computern wie PDAs und Organizer sowie Smartphones gedacht. Ursprünglich war diese Edition mit dem Hintergrund der geringen Nutzung von Speicher und reduzierter Rechenleistung realisiert worden. Mittlerweile haben jedoch auch moderne Geräte erheblich mehr Rechen-Power und höheren Speicher, so dass auch eine normale *Java Virtual Machine* (JVM) ablaufen kann.

Im Folgenden sind zunächst die wichtigsten Elemente der JDK aufgeführt.

Java Compiler. Der Java Compiler javac dient der Übersetzung des Java-Programms. Dabei wird ein plattformunabhängiger Zwischencode, der sogenannte Bytecode, erzeugt. Dieser wird in einer Datei, die denselben Namen wie die Quelltextdatei - jedoch mit der Datenerweiterung \*.class - besitzt,

gespeichert. Dieser Bytecode ist plattformunabhängig, weil er sich für die verschiedenen Rechnerplattformen beispielsweise Windows, Solaris oder Linux nicht unterscheidet. Allerdings ist dieser Zwischencode auch nicht direkt auf dem Zielsystem ausführbar.

Java Interpreter. Der Java Interpreter `java` realisiert die Ausführung des vom Compiler erzeugten Bytecodes und ist somit plattformabhängig. Es existieren somit für viele Plattformen entsprechende Interpreter. Dabei wird ein Java-Interpreter auch als Virtual Machine (JVM) bezeichnet.

Java Archiver. Das Dienstprogramm `jar` (Java-Archiv) ist über die Kommandozeile verfügbar und realisiert verschiedene Optionen, um Archive zu erzeugen, diese zu extrahieren und anzuschauen. Allerdings gibt es auch mit den in der JDK mitgelieferten Bibliotheken ein Application Programming Interface (API) mit der alles so programmiert werden kann, wie es auch das Dienstprogramm selbst leistet. Der Vorteil der Bündelung in einem Archiv ist der, dass Entwickler ihre Klassen- und Quellcode-Dateien komfortabel in einer einzigen ausführbaren Datei bündeln können. Ein weiterer Gesichtspunkt ist, dass die Betriebssysteme beispielsweise Windows und Mac OS X standardmäßig mit der Dateiendung `.jar` das *Java Runtime Environment* (JRE) verbunden haben, so dass ein Doppelklick auf einer Jar-Datei das Programm gleich startet.

Java Dokumentation. Das Java Tool `javadoc` erzeugt anhand spezieller Dokumentationskommentare im Java-Quellcode eine externe HTML-Datei für jede öffentliche Klasse. Zudem werden weitere Informationen wie eine Übersicht aller Klassen, Ausnahmen, Methoden und Schnittstellen in einer separaten Datei generiert. Dabei beachtet `javadoc` streng den in der objektorientierten Welt wichtigen Gesichtspunkt der Sichtbarkeit, und nimmt standardmäßig nur öffentliche Dinge in die Dokumentation auf.

Monitoringprogramme. Darunter sind seit der Versionen Java 5 und Java 6 Programme zusammengefasst, die gestartete Java-Programme auflisten und gezielte Anfragen an diese zur Laufzeit ermöglichen. Im Einzelnen sind das die Tools:

- Java Virtual Machine Process Status Tool (`jps`) liefert alle laufenden Java-Programme,
- Java Virtual Machine Statistics Monitoring Tool (`jstat`) ermöglicht Performance-Statistiken,
- Java Memory Map (`jmap`) listet alle Exemplare von Java-Objekten und den durch diese belegten Speicher auf und
- mit Hilfe des Stack-Trace-Tools (`jstack`) können laufende Threads zusammen mit Informationen über den durch Monitore erzwungenen Wartezustand angezeigt werden.

Zudem wird eine umfangreiche Sammlung von Bibliotheken, Komponenten und Beispielprogrammen bei einem Download der JDK zur Verfügung gestellt.

<http://java.sun.com>, <http://eclipse.org>

## JFC, Java foundation class

Zu den *Java Foundation Classes* (JFC) gehören alle Java-APIs, die zur Entwicklung von grafischen Benutzeroberflächen (GUIs) und sonstigen grafischen Desktop-Programmen benötigt werden. Insbesondere gehören dazu: das Abstract Windowing Toolkit (AWT), die *Swing* GUI-Komponenten, *Java 2D API* für zweidimensionale Grafikverarbeitung sowie die Bereiche Internationalisierung und Accessibility.

Die Java Foundation Classes bilden das Äquivalent zu den Framework Class Libraries (FCL) von .NET.

<http://java.sun.com/products/jfc/index.jsp>

**JHTML, Java HTML**

Mit JHTML (*Java* within Hypertext Language) wird eine Technik zur direkten Einbettung von Java-Code in eine Webseite - diese erstellt in HTML-Code - umschrieben. Dadurch werden die grundsätzlich statischen HTML-Dokumente um die dynamischen Aspekte der Java-Servlet-Technologie ergänzt. Dafür verantwortlich ist ein Servlet - das sogenannte PageCompileServlet, welches zur Übersetzung des Java-Codes den Java-Compiler aufruft, der letztendlich den eingebetteten Java-Code übersetzt. Die Verwendung von JHTML ist vergleichbar mit den Techniken, mit denen Microsofts Active Server Pages (ASP) oder die Scriptsprache PHP die Generierung dynamischer Aspekte in Webseiten ermöglichen. Eine alternativ angewendete Technik in Zusammenhang mit Java sind *Java Server Pages* (JSP).

JHTML ist ein einfacher Weg, dynamische Inhalte in statische HTML-Seiten zu integrieren und reduziert sich auf wenige Einzelschritte:

- Erstellung der HTML-Seiten mit einem z.B. einfachen Editor.
- Integration des Java-Codes mit entsprechender Kennzeichnung an beliebiger Stelle.
- Ablegen des HTML-/Java-Codes in einer Datei mit Endung "jhtml" auf dem Web Server.

Wenn ein Web-Client, im Allgemeinen der Browser, nun die Webseite anfordert, erkennt der Webserver diese jhtml-Datei und leitet den Code weiter zu einem speziellen Java-Programm - dem sogenannten PageCompile-Servlet. Dieses Servlet initiiert den Java-Compiler, der den Java-Code in ein Servlet übersetzt, welches dann die dynamischen Informationen generiert und der HTML-Seite hinzufügt, und diese schlussendlich dem Client zur Verfügung stellt.

Die JHTML-Technik bietet zum Beispiel auch die Möglichkeit, mit Hilfe des eingebetteten Java-Codes und der Nutzung der *Java Database Connectivity* (JDBC) auf externe, verteilte Datenbanken zu zugreifen. JHTML erfordert die Installation eines Java-Compilers auf dem Webserver. JHTML ist Bestandteil des Java Webserver Application Program Interface.

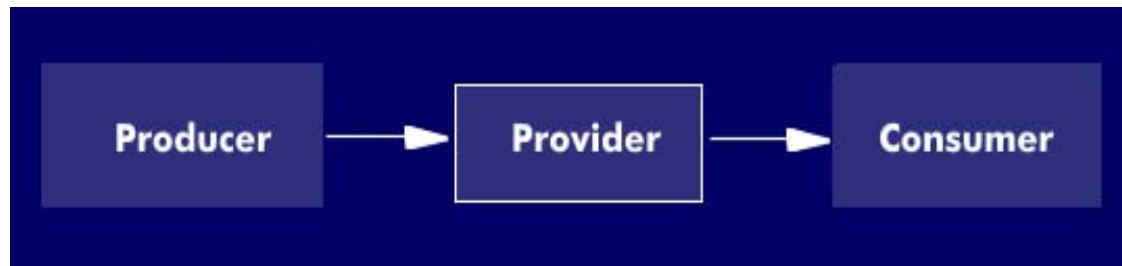
*JHTML-Programmierbeispiel*

```
<html>
<head>
<title> Demo JHTML </title>
</head>
<body>
<h2>Hello World</p2>
<p>Hello, world.</p>
<java>
Date date = new Date();
out.println("<p>The date is: "date.toString() "</p>");
</java>
</body>
</html>
```

**JMS, Java message service**

Der *Java Message Service* (JMS) definiert einen offenen Standard für eine einheitliche Java-Schnittstelle hinsichtlich Message Oriented Middleware (auch MOM oder nachrichtenorientierte Middleware). Im Sinne von MOM wird eine Message-Service-Implementierung mit ihren ggf. weiteren Funktionen wie Sicherheitsfunktionen, Load Balancing oder Werkzeuge zur Administration auch als JMS-Provider bezeichnet. Die Nutzer dieses Dienstes werden allgemein Client oder auch Producer und Consumer genannt. Producer und Consumer kommunizieren nicht direkt sondern über den Provider miteinander. Der sendende Client ist in Beziehung zum Provider immer asynchron, währenddessen der empfangende Client sowohl asynchron als auch synchron agieren kann. Vorteilhaft ist in jedem Fall, dass der JMS so unterschiedliche Plattformen einheitlich miteinander verbinden kann. Die JMS API unterstützt im Wesentlichen zwei Verfahren des Messagings - das Point-to-Point- sowie das Publish-and-Subscribe-Verfahren. Diese Verfahren werden auch als Messaging Domains bezeichnet. JMS wurde von Sun Microsystems entwickelt und ist Bestandteil der Java Enterprise Edition (JEE).

Die JMS-Spezifikation ist sehr umfangreich. Dies liegt zu einem gewissen Teil auch daran, dass zur Zeit der Erstellung dieser Spezifikation die Struktur von MOM-Systemen sehr uneinheitlich war, da diese bereits eine erhebliche Zeit vorher existierten. Die aktuelle Spezifikation ist in vollständigem Umfang unter dem u.g. Link verfügbar.



*JMS-Clients und Provider*

Die Abbildung zeigt den Provider mit seinen Clients - einen Producer, der die Nachricht generiert und sendet - sowie - einen Consumer, der die Nachricht empfängt und entsprechend agiert - in einer prinzipiellen Grundanordnung.

JMS unterstützt die folgenden Nachrichtenmodelle:

Point-to-Point (PTP). Hierbei wird die Nachricht - die genau einen Empfänger

hat - an eine Warteschlange (Queue) gesendet. Der Sender wird als Producer, der Empfänger als Consumer bezeichnet. Beide sind zeitlich völlig unabhängig voneinander. Die Queue dient der Speicherung persistenter Nachrichten, die entweder vom Consumer gelesen werden oder in Abhängigkeit eines Zeit-Limits ungültig werden können. Anwendung findet dieses Modell vor allem dort, wo eine zeitliche Entkopplung von Producer und Consumer möglich oder notwendig ist.

Publish-and-Subscribe (Pub/Sub). Analog wird in diesem Modell der Producer zum Publisher und der Consumer zum Subscriber. Die Nachrichten werden sogenannten Themen (Topics) zugeordnet. Viele Producer publizieren (publish) ihre Nachrichten zu einem Topic währenddessen wiederum beliebig viele Consumer die Nachrichten von dem Topic abonnieren (subscribe). Bei mehreren Consumern spricht man auch von multicast. Dieses Modell findet Anwendung bei Systemen, bei denen Nachrichten in Echtzeit zum Beispiel RSS-Feeds oder Börsendienste verteilt werden müssen.

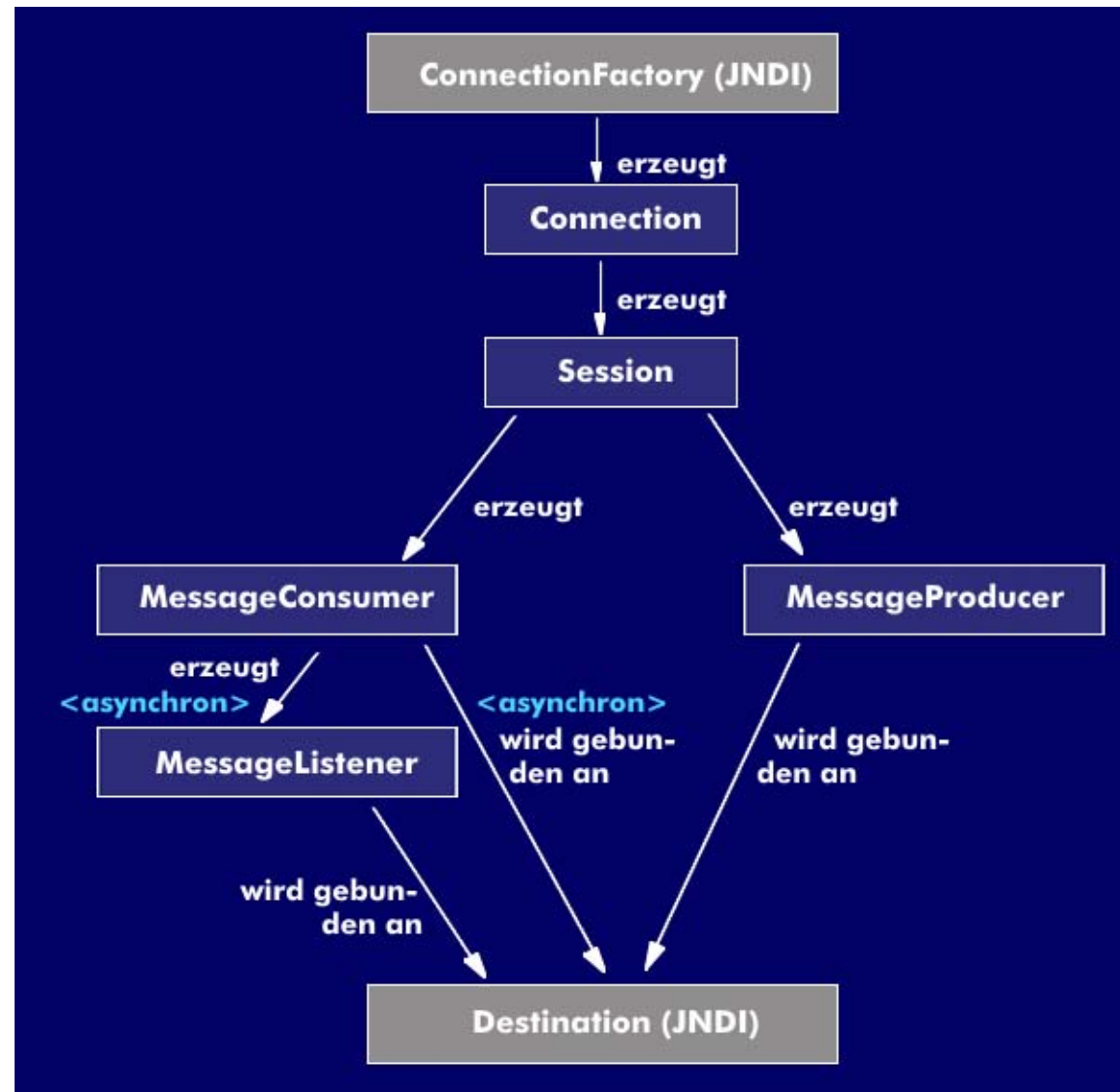
Eine JMS-Applikation kann nun eines dieser Nachrichtenmodelle verwenden oder auch Kombinationen dieser sogenannten Message Domains. In jedem Fall setzt sich die Architektur einer JMS Applikation aus den folgenden Elementen zusammen:

- Mit JMS-Clients werden die Java-Programme bezeichnet, die Messages senden und empfangen.
- Nicht-JMS-Clients (legacy clients) nutzen die API des Message Systems aber nicht JMS. Ein Message System unterstützt in der Regel beide Formen von Clients.
- Mit Messages werden die Nachrichten bezeichnet, die den Austausch von Informationen zwischen den Clients ermöglichen.
- JMS-Provider definieren ein Message System, welches JMS zusätzlich zu administrativen Funktionen implementiert.
- JMS-Objekte werden als administrative Objekte bezeichnet. Diese werden vom JMS-Provider administriert und stellen den JMS-Clients Verbindungsinformationen zur Verfügung.

JMS berücksichtigt nun, dass sich JMS-Provider u.U. hinsichtlich ihrer Konfiguration oder auch Technologie erheblich unterscheiden. Diese Eigenschaften werden auch als proprietär bezeichnet, und das Ziel ist die Portabilität von JMS-Clients und eingesetztem JMS-Provider. Dies wird durch die vom JMS-Provider erzeugten administrativen Objekte sichergestellt, von denen es zwei verschiedene Ausführungen gibt:

- ConnectionFactory - ein Objekt zum Aufbau einer Verbindung durch den JMS-Client.
- Destination - ein Objekt, welches Quelle und Ziel der Nachrichten definiert. Das Objekt bildet dabei eine Queue für die Nachrichten ab.

Die administrierten Objekte werden vom JMS-Provider ein *JNDI* (Java Naming and Directory Interface) eingetragen. Dadurch sind sie jedem JMS-Client auf portable Weise verfügbar, und auf die Objekte kann über die JMS API zugegriffen werden.



Allgemeines JMS-Modell

Die Abbildung verdeutlicht den Zusammenhang zwischen Verbindungsaufbau und Kommunikation über JMS. Zunächst wird dazu eine Verbindung (Connection) zwischen JMS-Client und JMS-Provider aufgebaut. Der JMS-Standard definiert dazu die Objekte Connection, Session, Message-Producer und MessageConsumer, über die ein JMS-Client die Dienste seines JMS-Providers erreichen kann. Die geeignete Implementierung der Schnittstellen ist Aufgabe des JMS-Providers, wobei der Ablauf je nach gewähltem Nachrichtenmodell unterschiedlich sein kann. Für einen Verbindungsaufbau und die Kommunikation muss ein JMS-Client die folgenden Schritte generieren:

- Über den Namensdienst (JNDI) die administrativen Objekte (ConnectionFactory und Destination) abfordern.
- ConnectionFactory ist für die

Verbindung zuständig, Destination implementiert die Queue des Providers.

- Es wird eine Session zur Steuerung der Kommunikation eröffnet, der MessageProducer wird mit dem Queue-Objekt verbunden.

Beim Senden einer Nachricht muss folgender Ablauf eingehalten werden: Die vom JMS-Client generierte Nachricht wird an den MessageProducer zur Ablage in der Queue übergeben.

Beim Empfang einer Nachricht muss unterschieden werden in: Asynchronen Empfang: Dazu installiert der JMS-Client einen MessageListener, der diesen informiert, wenn eine Nachricht für ihn beim JMS-Provider vorliegt, und synchronem Empfang: der JMS-Client wartet solange, bis der MessageConsumer eine Nachricht für ihn empfangen hat, was zu einer Blockade des JMS-Clients führen kann.

JMS ist nicht Multithreading fähig aber mit Einschränkungen unterstützt das Design von JMS mehrere User. So unterstützen die JMS-Objekte Destination, ConnectionFactory und Connection sogenannte Concurrent Uses.

<http://java.sun.com/products/jms/>, <http://openjms.sourceforge.net/>

## JMX, Java management extension

Die JMX-Technologie ermöglicht die Verwaltung von *Java*-Programmen, während diese ausgeführt werden. Dazu gehört vor allem die Überwachung des aktuellen Status, bei entsprechend entworfenen Anwendungen aber auch die Umkonfiguration. Diese Fähigkeit hat eine ganz besondere Bedeutung in Umgebungen, in denen viele *Java*-Programme zeitgleich ausgeführt werden und nicht nach Belieben gestoppt und neu gestartet werden dürfen.

Das funktioniert mit der Java Management Extension (JMX) allerdings nur auf einem lokalen Computer, um auch entfernt ausgeführte Programme zu verwalten gibt es die JMX Remote API.

<http://java.sun.com/products/JavaManagement/index.jsp>

## JNDI, Java naming and directory interface

Durch das *Java* Naming and Directory Interface (JNDI) haben *Java*-Programme Zugriff auf Namens- und Verzeichnisdienste wie beispielsweise Active Directory Service (ADS). Diese Verzeichnisse werden oft genutzt, um bestimmte Daten innerhalb einer Organisation zu verwalten und verschiedenste Dienste daran anbinden zu können. Oft werden beispielsweise Adressbücher und Telefonlisten in Unternehmen so zur Verfügung gestellt, damit aus anderen Anwendungsprogrammen wie Textverarbeitung oder E-Mail daraus zugegriffen werden kann. Auch im Rahmen der Benutzerauthentifizierung (beispielsweise über JAAS) spielen Verzeichnisse oft eine Rolle.

## JPDA, Java platform debugger architecture

Die *Java* Platform Debugger Architecture (JPDA) ist eine Architektur, die die Entwicklung von Debuggern für *J2SE* ermöglicht. Durch Offenlegung dieser Architektur ist es jedem Softwarehersteller möglich, Debugger für *Java*-Programme zu entwickeln.

## jQuery

jQuery beschreibt ein plattformunabhängiges Framework für die Scriptsprache *JavaScript*, das als Open-Source-Software zur Verfügung steht. Dabei realisiert jQuery konkret eine umfangreiche Klassenbibliothek, die den Zugriff auf das Document Object Model (DOM) vereinfachen. Die von John Reisig entwickelte Bibliothek liegt im Dezember 2009 in der Version 1.3.4 vor. Neben einer allgemein großen Verbreitung wird jQuery aufgrund seiner performanten Codierung u.a. sowohl in der Entwicklungsumgebung Visual Studio als auch der Web-Runtime-Plattform von Nokia eingesetzt. Die Verwendung von jQuery reduziert den Codeaufwand von *JavaScript*-Anwendungen erheblich.

Durch die von jQuery zur Verfügung gestellten Funktionen zur Manipulation und Navigation des DOMs wird die Durchsuchung und Manipulation, die Behandlung von Events, die Interaktion mit Ajax und die Animation von HTML-Seiten erheblich vereinfacht.

jQuery unterstützt dies durch eine sogenannte "Selektor" API (Programming Application Interface), die die Abfrage von HTML-Elementen mit Hilfe von Selektoren erlaubt, um dann entsprechende Befehle (commands) auf diese anzuwenden. Eine Eigenschaft der Commands ist es, dass diese verkettet werden können. Diese Verkettung von Aufrufen bezeichnet man auch als Invocation Chaining, so dass das Resultat eines Commands auch an andere, folgende Commands weitergegeben werden kann. jQuery beinhaltet auch eine Reihe von APIs für die Animation von HTML-Seiten.

jQuery wird von <http://docs.jquery.com/> geladen und per Script-Tag in die HTML-Dokumente eingebunden.

```
<script type="text/javascript"src="jquery.js"></script>
```

Die grundlegende Funktion von jQuery ist die Funktion `jQuery()` oder auch mit verringertem Schreibaufwand `$()`, deren Verhalten abhängig ist von den jeweils gesetzten Parametern. Dabei fasst (sammelt) jede `$()`-Funktion einen oder mehrere Knoten eines DOM-Baumes zusammen. In der einfachsten Form wird dann nur ein Ausdruck übergeben - meistens ein CSS-Selektor -, der alle passenden Elemente im Dokument findet.

Die Verwendung von jQuery hat für die Entwicklung von JavaScript-Anwendungen die folgenden Vorteile:

- der Aufwand an Code und Zeit reduziert sich erheblich,
- es ist einfach zu lernen und dabei standardkonform,
- das Warten auf das Laden des vollständigen Dokuments - mit allen Bildern, Stylesheets und sonstigen Elementen - entfällt dadurch, dass das DOM mit Laden des jeweiligen Elements sofort zur Verfügung steht,
- JavaScript wird um zahlreichen Funktionen wie `map`, `merge`, `trim`, `each`, `extend` und `grep` erweitert.

<http://jquery.com>

## **JRE, Java runtime environment** *Java Laufzeitumgebung*

Die *Java Runtime Environment* (JRE) ist ein Softwarepaket, das benötigt wird um Java-Programme auszuführen. Es enthält im Wesentlichen eine *Java Virtual Machine* (JVM) und die grundlegenden zur Ausführung benötigten Bibliotheken. Zur Entwicklung benötigte Werkzeuge wie der Compiler sind nicht Bestandteil der JRE sondern des *Java Development Kits* (JDK).

## **JRuby**

JRuby umschreibt einen Ruby-Interpreter für die *Java Virtual Machine* (JVM). JRuby ist ein vollständig in *Java* implementierter Interpreter, und ermöglicht - vergleichbar u.a. mit *Scala*, *Groovy*, *BeanShell*, *Jython*, *Jacl* - die Nutzung der JVM auf Basis der Programmiersprache *Ruby*. Der Hintergrund von JRuby ist die Interoperabilität der beiden Sprachen *Ruby* und *Java*. Damit sind dann die Konzepte der objektorientierten, prozeduralen und funktionalen Paradigmen sowie der nebenläufigen Programmierung auf der JVM ausführbar. Der Sprachumfang von *Ruby* wird von JRuby nahezu vollständig implementiert und unterliegt einer ständigen Weiterentwicklung durch das Team *Charles Nutter* und *Thomas Enebo*. Diese stellen bezüglich JRuby auch immer wieder umfangreiche Informationen zur Verfügung. So wurde in 2009 eine Version von JRuby ausführbar auf der Software-Plattform für mobile Geräte - *Android* - vorgestellt. Das plattformunabhängige JRuby liegt im Dezember 2009 in der Version 1.4 vor. Zusätzlich ist ein Compiler verfügbar, der *Ruby*-Klassen (Version 1.8) in *Java*-Klassen übersetzt. JRuby realisiert als Laufzeit-Interpreter die Möglichkeit, die Eigenschaften der Programmiersprache *Ruby* auf der JVM abzubilden. Dadurch können die umfangreichen *Java*-Bibliotheken in den *Ruby*-Programmcode eingebettet werden. Dieser Mechanismus wird auch als "subclassing java" umschrieben. Durch die Verwendung von JRuby können allerdings auch *Java*-Programme in *Ruby* implementierte Programme aufrufen. Somit ist durch JRuby die Gelegenheit gegeben, zusätzlich zur Integration bestehender *Java*-Anwendungen und deren Bibliotheken, weitere transparente *Ruby-on-Rails* Anwendungen zu entwickeln. Ein klarer Vorteil von JRuby ist, dass der Programmcode bei gleicher Funktionalität im Verhältnis zu *Java* reduziert ist. Entsprechende Messungen der Performance des ausführbaren Programmcodes bestätigen JRuby erhebliche Vorteile im Vergleich zur Verwendung der *MRI*-Umgebung (*Matz Ruby Interpreter*). JRuby ist damit auch eine ausgezeichnete Laufzeitumgebung für *Ruby-on-Rails*-Anwendungen. In diesem

Zusammenhang ist aber ebenso auf die JVM hinzuweisen, die mit der HotSpot-Technologie unter aktuellen Gesichtspunkten eine sehr hohe Performance bei der Ausführung von JRuby-Code erzielt.

<http://jruby.org>

## **JSP, Java server pages**

*Java Server Pages (JSP)* ist eine Entwicklungssprache für Web-Anwendungen. Mit ihr können Webentwickler plattformübergreifende Programme erstellen und darin HTML- und XML-Tags und so genannte Java-Scriptlets verwenden. Letztere werden vom Server und nicht vom Web-Browser ausgeführt und können dynamische Webseiten erstellen. Darüber hinaus unterstützen die Java Server Pages die Integration verschiedenster Inhalte, dadurch kann man beispielsweise Datenbankinhalte auf den Webseiten darstellen oder auch Dateien mit multimedialem Inhalt. Ebenso können die JSP-Pages, die vollständig mit Java programmiert sind, mit *JavaBeans* angereichert werden.

## **JSR, Java specification request**

Die *Java Specification Request (JSR)* ist ein Dokument, in dem eine mögliche Weiterentwicklung der Programmiersprache Java beschrieben wird. JSRs werden vom *Java Community Process (JCP)* erstellt und veröffentlicht mit dem Ziel, daraus eine tatsächliche Änderung an der Sprache erwachsen zu lassen. <http://www.jcp.org>, <http://www.jcp.org/en/jsr/all>

## **JVM, Java virtual machine** *Java virtuelle Maschine*

*Java Virtual Machine (JVM)* ist eine Software, die zur Ausführung von Java-Programmen benötigt wird und Bestandteil der *Java Runtime Environment (JRE)* ist. Im Gegensatz zu Java-Programmen selbst ist die JVM nicht plattformunabhängig, sondern ein natives Programm für die jeweilige Zielplattform. Auf jedem System, für das eine den Spezifikationen entsprechende JVM verfügbar ist, können Java-Programme ausgeführt werden.

Nachdem der Quelltext in Bytecode umgewandelt worden ist, kann dieser der JVM übergeben werden. Diese sucht den Einstiegspunkt des Programms (im Fall von Java die `main()`-Methode) und beginnt dort mit der Ausführung. Dazu wird der auszuführende Programmblock zunächst für das Zielsystem kompiliert und dann aufgerufen. Die Kompilierung erfolgt als *Just In-time Compilation (JIT-Kompilation)*.

Als gravierender Nachteil dieser Vorgehensweise wird im Allgemeinen der erhöhte Ressourcenbedarf ins Feld geführt, der auf Grund der auszuführenden JVM und ständigen Kompilierungsvorgänge nicht zu vermeiden ist. Optimierungen in diesem Bereich und das stetige Leistungswachstum der Hardware lassen es jedoch in den meisten Anwendungsgebieten immer weiter in den Hintergrund treten.

Die zweite Hauptaufgabe der JVM, vereinfacht die Programmierung im Vergleich zu älteren Sprachen wie insbesondere C++ erheblich: die Verwaltung der verwendeten Speicherbereiche. Java-Programmierer werden vollständig von der Aufgabe befreit, Speicherbereiche explizit zu reservieren und wieder freizugeben. Neben kürzeren Entwicklungszeiten erhält man so stabilere Software, die weniger anfällig für Speicherlecks und Sicherheitslücken wie Buffer Overflows ist. Die wichtigste Funktion in diesem Zusammenhang ist die Garbage Collection.

Die automatische Speicherverwaltung arbeitet zwar zuverlässiger und sicherer, aber nicht so effizient wie eine manuell (und fehlerfrei) implementierte. Deshalb gibt es durchaus Anwendungsfälle, in denen Java als Programmiersprache weniger gut geeignet ist als beispielsweise C++.

**Swing** Swing ist eine Programmierschnittstelle (API), die die Programmierung von plattformunabhängigen GUIs mit *Java* ermöglicht. Es ist Bestandteil der *Java Foundation Classes* (JFC) und gehört somit zur *Java 2 Platform, Standard Edition* (J2SE).

Das ebenfalls zu den Java Foundation Classes (JFC) gehörende AWT-API ist die unverzichtbare Grundlage für Swing. Das Abstract Windowing Toolkit (AWT) benutzt die Bibliotheken des jeweiligen Ziel-Betriebssystems, um Fensterrahmen zu zeichnen und notwendige Aufgaben wie die Ereignisbehandlung abzuwickeln. Swing seinerseits zeichnet die eigentlichen GUI-Komponenten in die leeren Rahmen und ist darauf angewiesen, dass die Ereignisbehandlung funktioniert.

Swing folgt in der Architektur relativ streng dem MVC Entwurfsmuster, das die Stärken der objektorientierten Programmierung voll ausspielt. Gleichzeitig ist es aber auch relativ komplex und erfordert in vielen "leichtgewichtigen" Oberflächen verhältnismäßig viel Programmierarbeit. Deshalb gibt es viele Entwickler, die für die GUI-Entwicklung andere Programmierplattformen benutzen wie auf .NET oder andere Programmiersprachen wie Delphi mit den Bibliotheken VCL (Visual Components Library) und CLX (Component Library for Cross Platform Development) oder C++ mit der Bibliothek Qt. Für Java hat sich zudem noch die alternative GUI-Bibliothek Standard Widget Toolkit (SWT) etabliert.

<http://java.sun.com/products/jfc/index.jsp>

<http://eclipse.org/swt/>