

# **Skript Informatik B**

## **Objektorientierte Programmierung in Java**

**Sommersemester 2011**

**- Teil 2 -**

# Inhalt

0 Einleitung

1 Grundlegende objektorientierte Konzepte

2 Grundlagen der Software-Entwicklung

3 Wichtige objektorientierte Konzepte (Major Concepts)

... wird schrittweise erweitert

# Kapitel 2:

## Grundlagen der Software-Entwicklung

- 2.1 Die Programmiersprache Java
- 2.2 Vom Programm zur Maschine
- 2.3 Abstraktion von der Plattform
- 2.4 Vom Problem zum Programm
- 2.5 Programmiersprachenparadigmen

In diesem Kapitel werden die „Begleiterscheinungen“ der Programmierung im Überblick behandelt.

Wer Hintergrundwissen zu Java hat, kann einige Dinge in der Java Programmierung besser verstehen. Das gleiche gilt für das Wissen zur Verbindung zwischen Programm – Betriebssystem – Hardware.

Obwohl eine Programmiersprache, die Kenntnis der darunter liegenden Hardware prinzipiell überflüssig macht, stimmt das in der Realität nicht.

Ein Programmierer kann das Programmverhalten oft besser oder teilweise sogar erst dann verstehen, wenn er weiß, wie die Verbindung zur und Abarbeitung auf der Plattform aussieht. Die Themenbereiche, die dabei eine Rolle spielen sind: Programmiersprachenkonzepte, Betriebssysteme, Rechnerarchitektur und Rechnernetze (für Anwendungen, die in einem Netzwerk laufen sollen).

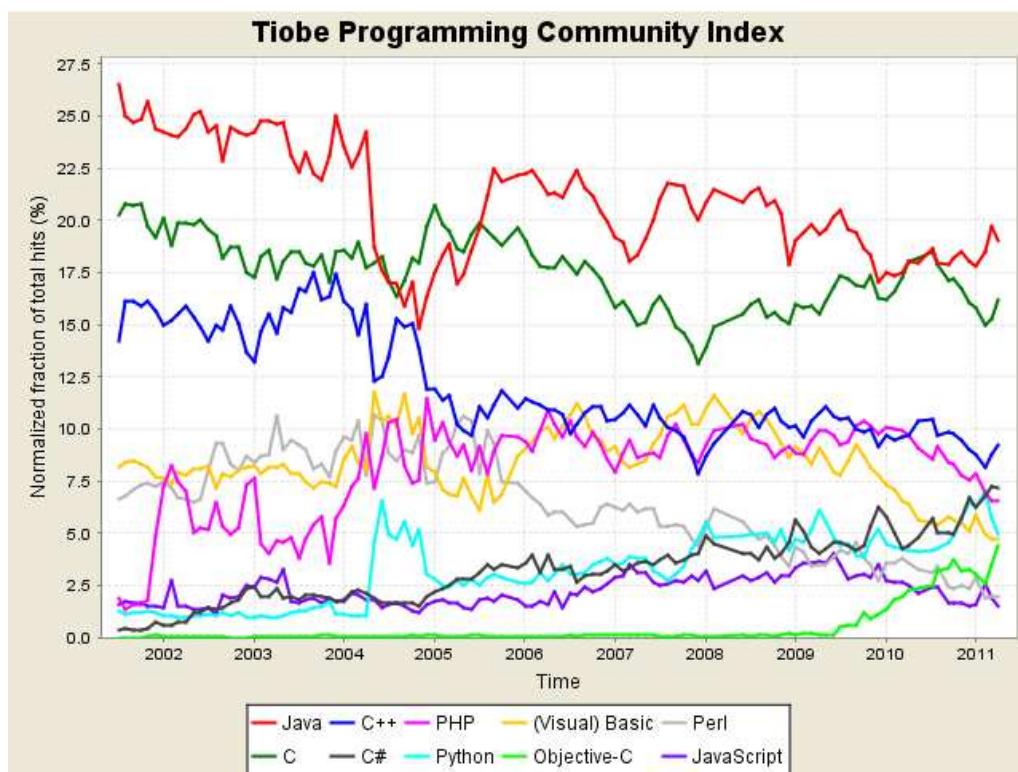
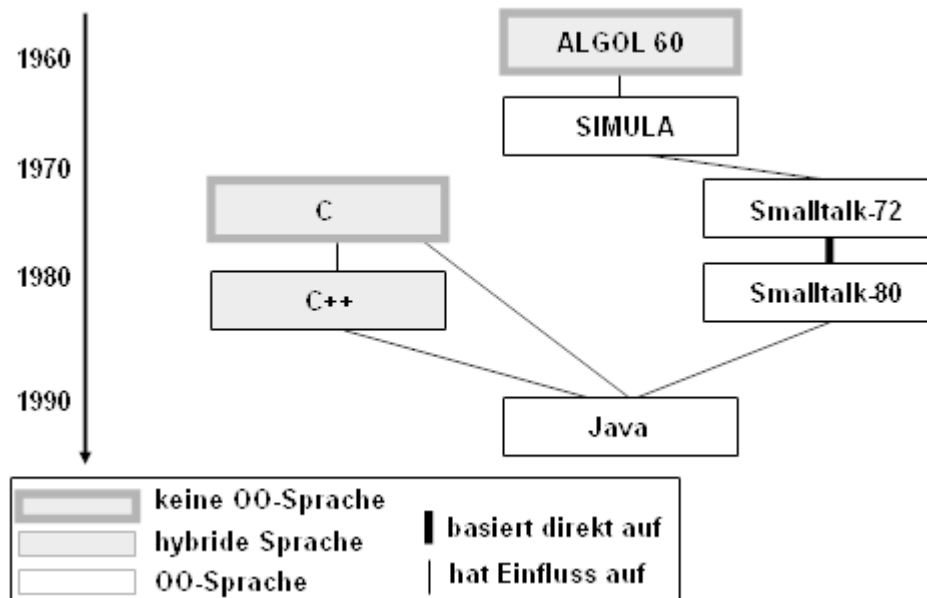
Eine Entwicklerin muss in aller Regel zunächst ein Problem verstehen und eine Lösung gestalten, bevor sie zur tatsächlichen Programmierung schreitet. Dazu wird Modellierungs-Know-how benötigt. Die Modellierung des Problems wird vor und begleitend zur Umsetzung betrieben. Die Themenbereiche, die hierbei eine Rolle spielen sind zentrale Arbeitsgebiete des Software Engineering: Modellierungsvorgehen, Erfahrungswerte in der Entwicklung (z.B. bewährte Entwurfslösungen und Architekturen), Modellierungssprachen (z.B. UML), Werkzeuge und Entwicklungsumgebungen, Projektmanagement zur Koordination vieler Entwickler, Verwaltung der Software-Versionen und der Änderungen (Konfigurationsmanagement), Qualitätssicherung (z.B. Fehlererkennung).

Soll die ausgedachte Lösung schließlich umgesetzt werden, muss spätestens dann eine geeignete Programmiersprache ausgewählt werden. Ein Programmierer braucht dazu einen Überblick, welche Sprache für das Problem bzw. die Lösung geeignet ist. Das muss nicht zwangsläufig immer Java bzw. eine objektorientierte Sprache sein. Das Wissen aus den Bereichen Programmiersprachenkonzepte, Compilerbau und Software Engineering spielen für die Umsetzung eine wesentliche Rolle.

Die Umsetzung in eine konkrete Implementierung geschieht in größeren Projekten immer Hand in Hand mit einer weiterführenden Modellierung.

## 2.1 Die Programmiersprache Java

### 2.1.1 Entwicklung der Programmiersprachen



Aus: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

## 2.1.2 Entwicklung der Programmiersprache Java

- Java Entwicklung:
  - 90iger Jahre: Programmiersprache für Consumer Geräte, Set-Top-Boxen, Netzwerk-Anwendungen
  - heute: hauptsächlich zur Realisierung Internet-basierter / verteilter Anwendungen
  - OpenJDK (<https://openjdk.dev.java.net/>) als Basis von Java 7: Endgültige Freigabe aller Bestandteile als Open Source-Lizenz GPL 2
- Breite Industrieunterstützung für Java  
Sun/Oracle, Apache Software Foundation, BEA, IBM, (Microsoft)
- Konkurrenten: C#, C++

Aktuelle Entwicklungen zum Thema Sun, Oracle, Java in den Nachrichten:

### SOFTWARE-ENTWICKLUNG

## Oracle kauft Sun: Deutsche Java-User sind gelassen

von Armin Barnitzke

23. APRIL 2009

**Java gilt als Suns Kronjuwel. Angesichts der Übernahme durch Oracle bleiben deutsche Java-Anwender aber gelassen. Das zeigt eine Umfrage unter Mitgliedern der Java User Group Stuttgart (JUGS). Die Ellison-Company engagiere sich bereits stark für Java und werde einen Teufel tun, diese Technologie aufs Spiel zu setzen, in dem sie diese als Druckmittel gegen SAP oder IBM einsetzt. Damit würde Oracle nur die Community gegen sich aufbringen – und Microsofts Dotnet in die Hände spielen. Mehr Fragezeichen sehen die Java User dagegen im Open-Source-Umfeld.**

Zudem: „Ein Einsatz von Java als Waffe gegen SAP und IBM würde Java insgesamt schwächen – und damit Oracle abhängig von Microsoft machen, die mit Dotnet eine Alternative haben, die damit hoffähig werden könnte“, bekräftigt Gerd Castan. „Das würde Oracle insgesamt also mehr schaden als nutzen.“

Aus der ehemaligen Computerzeitung: <http://www.computerzeitung.de/>

Markt 20.04.2009 - 19:25

Alexandra Kleijn

## Oracle kauft Sun: Was wird aus Java, MySQL und Openoffice?

Nach den wochenlangen Spekulationen, ob und zu welchen Konditionen IBM Sun kauft, ist es jetzt Datenbank-Koloss Oracle, der das angeschlagene Unternehmen übernimmt. Damit kommen (Open-)Solaris, Java, MySQL und OpenOffice unter neue Regie.

Java kommt bei dem Datenbank-Hersteller erklärtermaßen eine strategische Rolle zu, schließlich basiert Oracles Middleware Fusion auf Java – das Unternehmen selbst spricht von Java als der wichtigsten Software, die Oracle je eingekauft hat. Java steht mittlerweile unter GPL; und selbst wenn Oracle aus welchen Gründen auch immer die Weiterentwicklung von Java wieder schließen sollte, kann man getrost davon ausgehen, dass es genug Interessenten an einem freien Java gibt, um dessen Entwicklung sicherzustellen.

Aus: <http://www.heise.de/open/Oracle-kauft-Sun-Was-wird-aus-Java-MySQL-und-OpenOffice--/artikel/136431>

 heise online - news

News Autos iX c't

Home Archiv weitere Angebote

11.04.2010 13:52 Uhr

« Überblick 11.04.2010

### Java-Erfinder verlässt Oracle

James Gosling, Erfinder der Programmiersprache Java und bisher Chief Technology Officer von Oracles Client Software Group, hat das Unternehmen verlassen. In seinem [Weblog](#) bestätigt er Gerüchte, laut denen er bereits zum 2. April gekündigt habe. Demnach war er auch nicht wie anscheinend von einigen erwartet auf den [TechDays](#) des Sun Developer Network im russischen St. Petersburg anwesend. Es sei noch nicht klar, welche Arbeit er als nächstes angehen werde.

Gründe für sein Ausscheiden gibt Gosling nicht an. Im Januar, als die EU-Kommission die Übernahme von Sun durch Oracle [genehmigt hatte](#), [veröffentlichte](#) Gosling in seinem Weblog einen Abschiedsgruß an Sun. Im Februar hatte bereits der ehemalige Sun-CEO Jonathan Schwartz seinen [Rücktritt erklärt](#). Auch XML-Miterfinder Tim Bray hat Oracle bereits [verlassen](#). ([anw](#))

Aus: <http://heise-online.mobi/news/Java-Erfinder-verlaesst-Oracle-974931.html>

Oracle treibt aktuell die Weiterentwicklung von Java nicht mehr mit gleicher Energie voran wie ehemals Sun.

## 2.1.3 Entwurfsziele und Intention der Programmiersprache Java

### Java ist konzipiert als eine ...

- einfache und kleine
- objektorientierte
- dezentrale
- interpretierte
- stabil laufende
- architekturneutrale [plattformunabhängig]
- portierbare
- dynamische

**Sprache, die Hochgeschwindigkeitsanwendungen und Multithreading unterstützt** (zitiert nach SUN Microsystems).

Java ist eine *General Purpose* Sprache, d.h. sie kann in vielen Anwendungsgebieten eingesetzt werden.

## 2.1.4 Programmierung in Java

### Arten von Java-Programmen:

- Java-Applets
  - in Web-Seiten eingebettet
  - in der Laufzeit-Umgebung von Web-Browsern ausgeführt
  - aus Sicherheitsgründen eingeschränkte Rechte
- Java-Applikationen
  - von Kommandozeile oder als Anwendung gestartet
  - volle Rechte einer eigenständigen Anwendung
- Die Programmierung von Applets und Applikationen ist ähnlich zueinander.

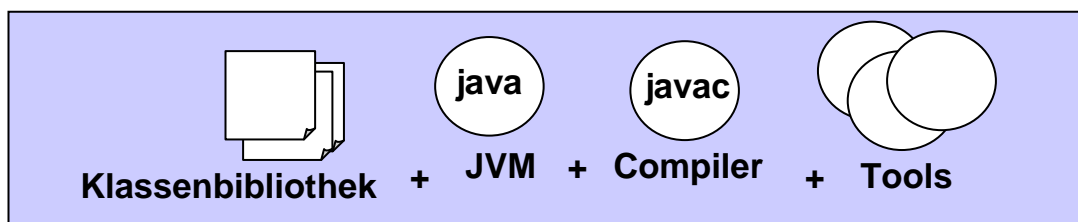
### Verschiedene Java Plattformen für verschiedene Aufgabenbereiche:

Je nach Anwendungsgebiet gibt es verschiedene Java-Plattformen:

- Java SE (Java Platform Standard Edition, früher J2SE): Die Systemumgebung für die Entwicklung und Ausführung von Java-Programmen. Mit dieser Umgebung arbeiten wir.
- Java ME (Java Platform, Micro Edition): Java-Umgebung für tragbare Computer wie PDAs, Organizer, Telefone. Mit wachsender Leistungsfähigkeit dieser Geräte und mit der Plattform Android von Google verliert diese Umgebung tendenziell an Bedeutung.
- Java Card: Standard für Java-ähnliche Programme auf Chipkarten (Smardcards). Für derart kleine Systeme wurde Java eingeschränkt und angepasst.
- Java EE (Java Platform, Enterprise Edition): Aufsatz für Java SE mit Paketen zur Entwicklung von Geschäftsapplikationen (Enterprise-Applikationen). Ergänzungen sind z.B. Enterprise JavaBeans (EJBs), Servlets, Web-Services.

### Bausteine und Werkzeuge zur Programmierung in Java SE:

- Java Development Kit (JDK)
  - kostenlose Java-Entwicklungsumgebung von Sun/Oracle



- zur Java-Entwicklung reichen prinzipiell ein Text-Editor und das JDK

- integrierte Entwicklungsumgebung (Integrated Development Environment = IDE)

↓  
kommerziell:

- Borland JBuilder, ...  
<http://www.embarcadero.com/products/jbuilder>

↓  
frei:

- Eclipse (IBM/Konsortium, 2001),
- NetBeans (Sun/Oracle),
- BlueJ ([www.bluej.org](http://www.bluej.org))

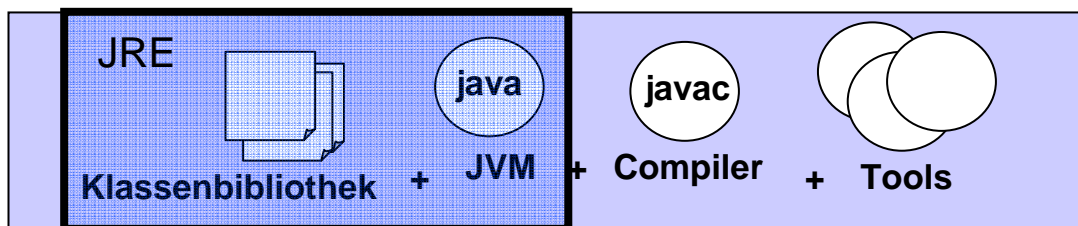
**Java Entwicklungswerkzeuge (Tools):**

- javac Java Compiler  
(javac Example.java)
- java Java-Interpreter (Java Application Launcher)  
(java Example)
- appletviewer Applet-Viewer  
(appletviewer Example.html)
- javadoc Dokumentationswerkzeug (API Documentation Generator)  
(javadoc Example.java)
- jdb Textmode-Debugger (Java Application Debugger)  
(jdb Example)
- javap Disassembler (Class File Disassembler)  
(javap Example)
- javah C Header and Stub File Generator  
(javah Example)
- jar JAR Archive Tool

**Weitere Details zu den Bausteinen in Java (v.a. Java SE):**

Java ist zum einen eine Programmiersprache, zum anderen steht Java für eine Menge von Standardbibliotheken und eine Laufzeitumgebung.

- Java Runtime Environment (JRE)
  - Untermenge des JDK



- (mindestens) benötigt, um Java Applikationen auszuführen
- darf ohne Lizenzgebühren eigenen Produkten beigelegt werden

- Java Virtual Machine (JVM)
  - Teil des JRE, der Bytecode (in den .class Dateien) interpretiert und ausführt
  - .class Dateien werden bei Bedarf geladen und interpretiert
  - oder "just in time" in Maschinensprache der Zielplattform übersetzt (JIT Compiler)

**Bemerkung:**

Die Bausteine sind selbst ebenfalls nur Anwendungen, die in einer Programmiersprache geschrieben sind. Die Spezifikation von Java gibt nur an, was an Funktionalität erbracht wird, aber nicht, wie es intern realisiert ist.

**Beispiele:**

Der Java Compiler in Eclipse ist selbst in Java realisiert.

Der Java Compiler Jikes (IBM) ist in C++ realisiert.

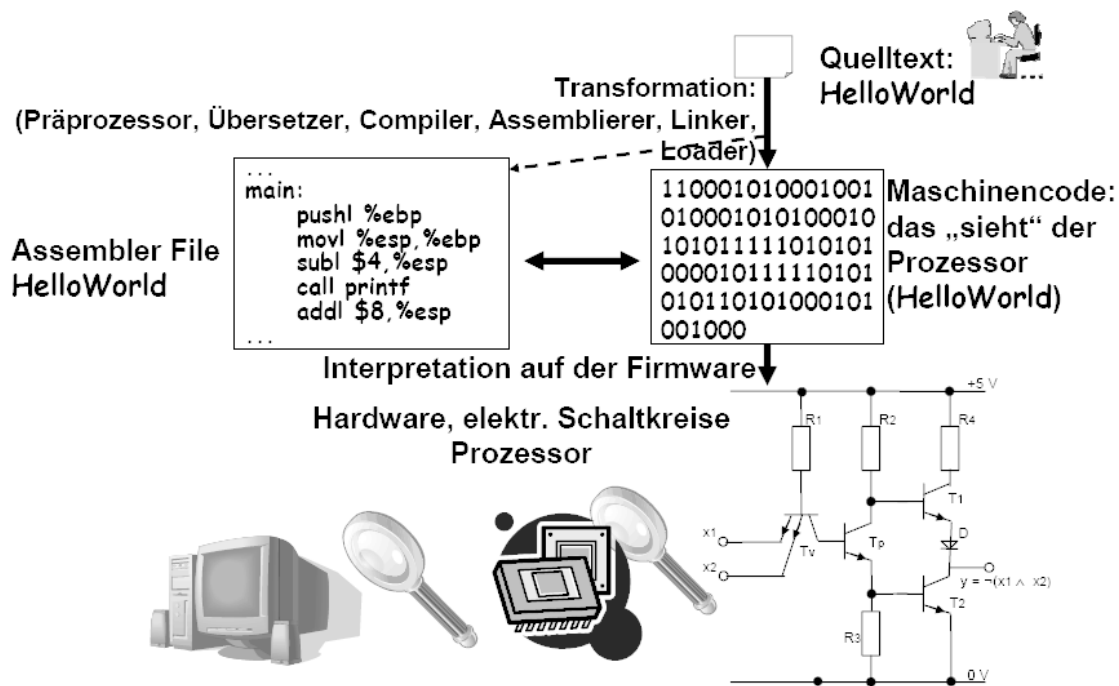
Die Java Virtual Machine ist (aus Geschwindigkeitsgründen) häufig in C++ implementiert.

Neben den Java Standardbibliotheken gibt es weitere kommerzielle oder quelloffene Bibliotheken (z.B. zum Schreiben von PDF-Dokumenten).



## 2.2 Vom Programm zur Maschine

### 2.2.1 Allgemeiner Ablauf der Programmverarbeitung

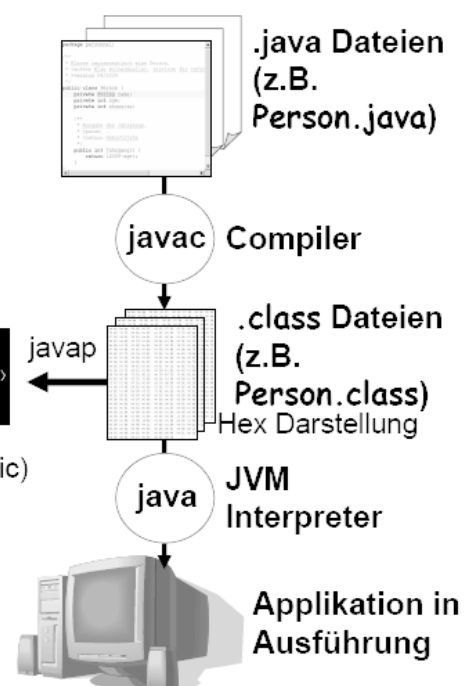


### 2.2.2 Ablauf der Programmverarbeitung am Beispiel Java

1) Programmierung: Entwickler schreibt Quelltext (Source Code)

2) Kompilierung: Übersetzer/Compiler übersetzt Quellcode in Binärcode (Java-Binärcode = Bytecode)

3) Java Virtual Machine interpretiert Bytecode, führt ihn aus

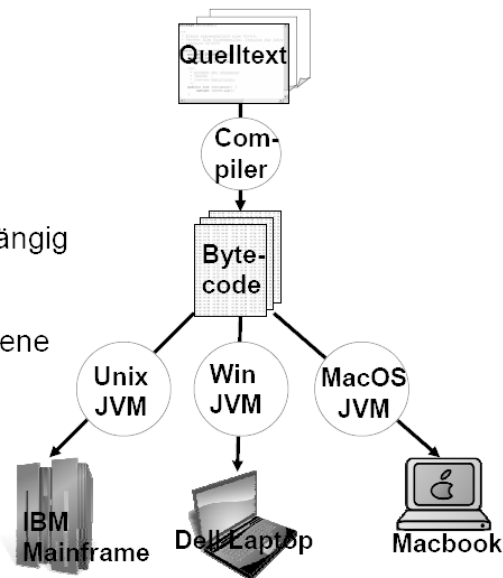


## Plattformunabhängigkeit:

- Bytecode ist plattformunabhängig

- JVM existieren für verschiedene Plattformen

→ kompilierte Programme auf verschiedenen Plattformen ausführbar



Leitgedanke: „Write once, run anywhere“ (WORA)

Java “bezahlt” die Plattformunabhängigkeit durch Einbußen in der Performance (Ausführungsgeschwindigkeit) und durch - gegenüber anderen Sprachen - mehr Bedarf an Arbeitsspeicher.

**Compilation:**

Quellprogramm

↓  
**Compiler**

↓  
**Zielprogramm  
(Maschinensprache)**

↓  
Ausführung



versus

**Interpretation:**

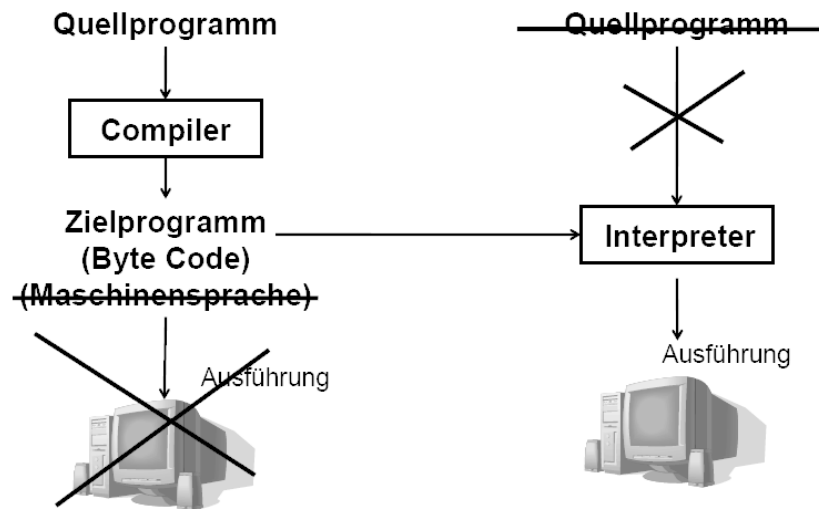
Quellprogramm

↓  
**Interpreter**

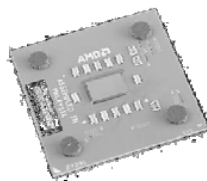
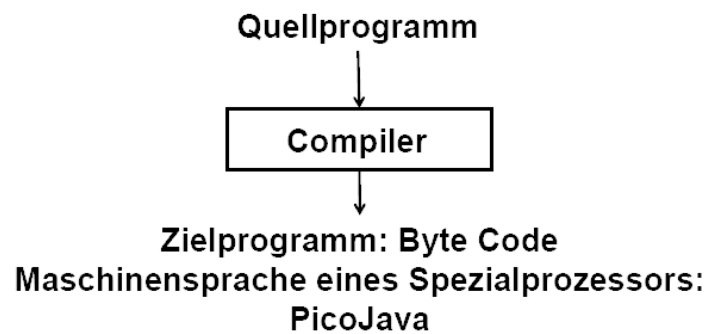
↓  
Ausführung



## Wie macht es Java?



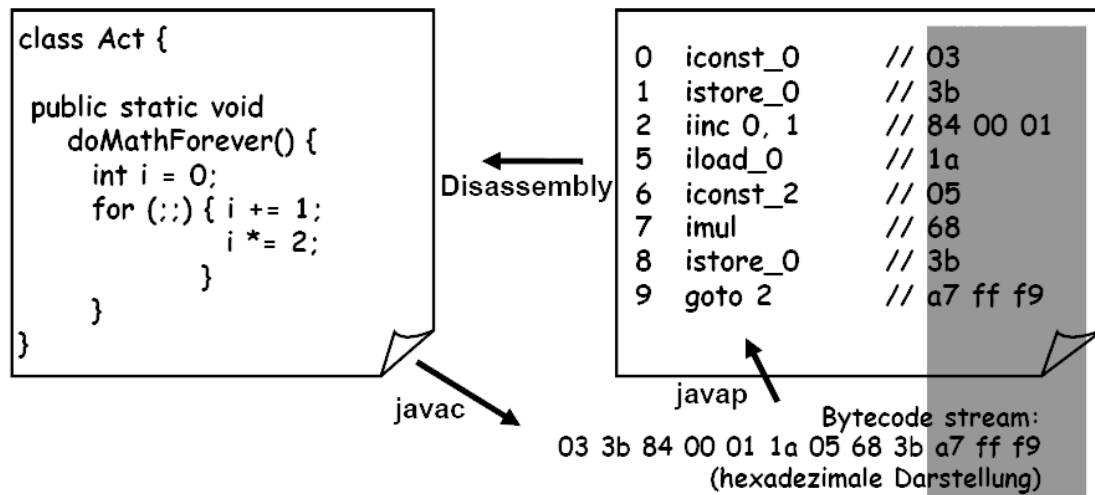
## Wie macht es Java (die Hardware-Alternative)?



„Java on a Chip“

## 2.2.3 Vom Programm auf die virtuelle Maschine und umgekehrt

[Bill Venners, Inside the Java Virtual Machine, 2000]



### Beispiel für Java Byte-Code:

```

// Bytecode stream: 03 3b 84 00 01 1a 05 68 3b a7 ff f9
// Disassembly:
iconst_0      // 03
istore_0      // 3b
iinc 0, 1     // 84 00 01
iload_0       // 1a
iconst_2      // 05
imul          // 68
istore_0      // 3b
goto -7       // a7 ff f9
  
```

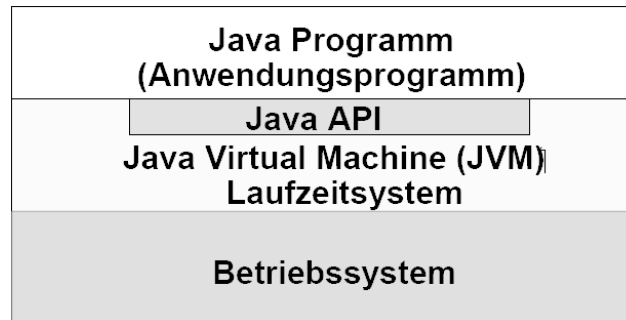
Bytecode zum Weiterlesen:

<http://www.javaworld.com/javaworld/jw-09-1996/jw-09-bytecodes.html>

## 2.3 Abstraktion von der Plattform

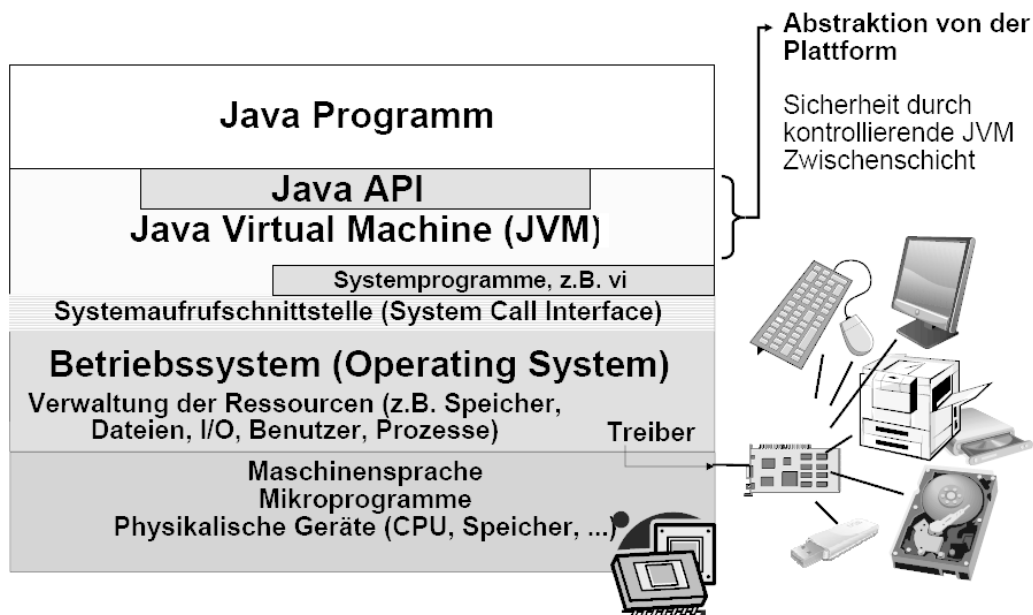
### 2.3.1 Abstraktion in Java

Eine Abstraktion von der Plattform wird in Java mit Hilfe einer Schichtenarchitektur und der JVM erreicht:

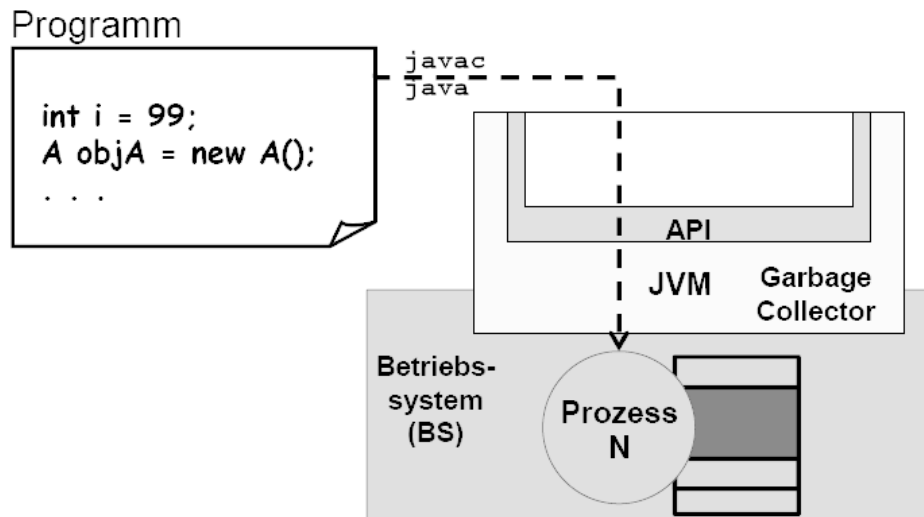


JVM zum Weiterlesen: Tim Lindholm, Frank Yellin: The Java™ Virtual Machine Specification, Second Edition, 1999

Online frei verfügbar: <http://java.sun.com/docs/books/jvms/>



Die JVM lädt die Klassen mit des Klassenladers (engl. Class Loader) von verschiedenen Lokationen – gegebenenfalls auch über das Netzwerk. Das ist ein Problem für die Sicherheit. Java hat daher ein Security-Modell: Ein *Verifier* sorgt für die Prüfung des Codes in Hinblick auf Struktur und Typsicherheit. Ein *Security Manager* überwacht die Zugriffe auf lokale Ressourcen.



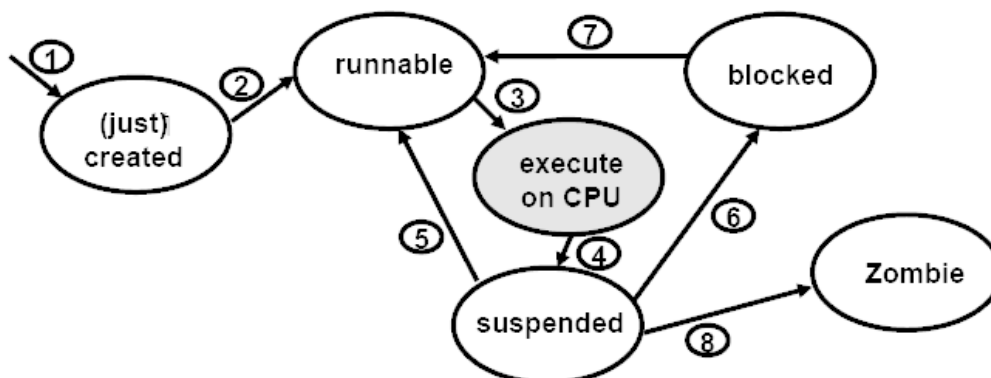
Weil Java plattformunabhängig entworfen wurde, ist es mit Java eher umständlich, maschinen- und plattformspezifische Anforderungen umzusetzen (z.B. Bildschirmfarbe setzen, CD auswerfen). Für eine solche hardwarenahe Systemprogrammierung sind andere Sprachen wie z.B. C++ geeigneter.

Sollen in Java systemnahe Eigenschaften angesprochen werden, geschieht dies in der Regel durch entsprechende Bibliotheken, über die eine Systemfunktion nativ gerufen werden kann. Die nativen Methoden sind Unterprogramme, die nicht in Java implementiert sind (häufig in C oder C++). Diese Bibliotheken müssen pro Plattform bzw. Geräteart neu bereitgestellt werden und auch aus Java heraus plattformspezifisch aufgerufen werden.

### 2.3.2 Systemressource “Prozess”

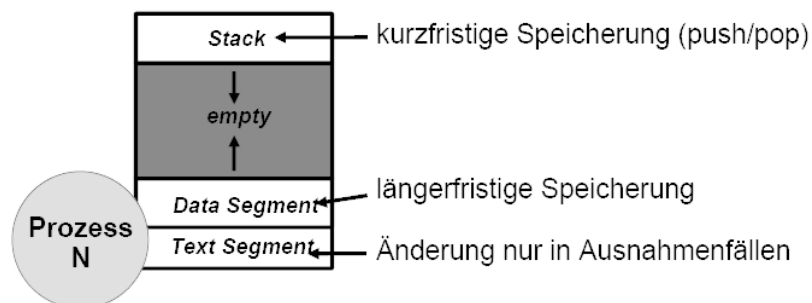
Prozesse sind eine Systemressource. Diese werden vom Betriebssystem verwaltet.

- Prozesse (Processes / Tasks): Sie sind die ausführbaren / ausgeführten Programme auf einem Rechner.
  - Multi-Processing vs. Single-Processing
  - Multi-User vs. Single-User
- Prozesszustände:

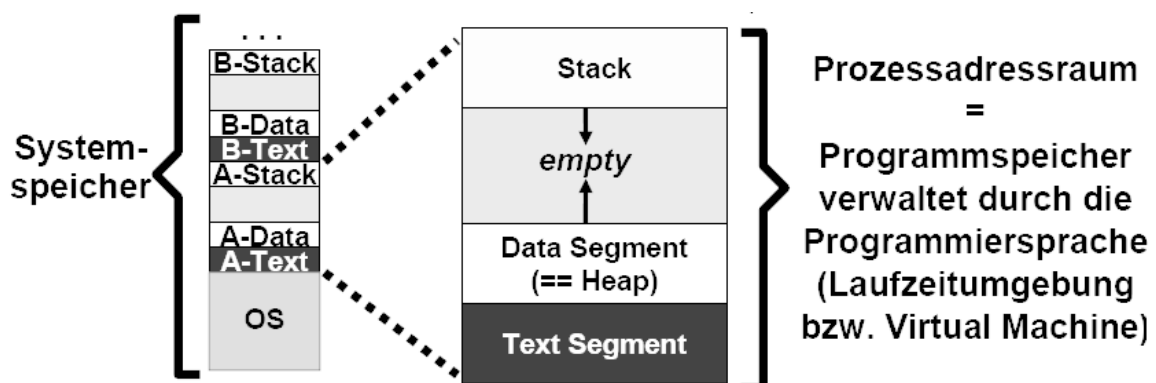


- Prozesse besitzen eigene Speicherbereiche (Adressräume) für Daten und Code.

- Prozesse besitzen Zugriffsrechte (engl. Access Rights) auf Ressourcen wie Dateien oder Netzwerkverbindungen.
- Prozesse haben Verwaltungsinformationen über ihren Zustand, z.B. Priorität und Ablaufzustand.
- Das Betriebssystem hält in einer Prozesstabelle (engl. Process Table) alle Prozesse gespeichert (Process ID, pid) und steuert den Ablauf der Prozesse.
- Einem Prozess ist ein Speicherbereich zugeordnet = Adressraum eines Prozesses
- Der Adressraum eines Prozesses (Speichermodell) besteht aus:  
Datensegment, Programmsegment und dem Stack (Stapelspeicher)



- Speicherbedarf ergibt sich für:
  - die Speichervariablen gemäß Datentyp
  - das Programm / den Programmtext
- Speicherverwaltung für Prozesse durch das Betriebssystem (UNIX Speichermodell):



Der Stack-Speicherbereich wächst nach unten, der Heap nach oben.

Treffen die beiden Speicherbereiche zusammen, wird das Betriebssystem aktiv: Es weist dem Prozess einen neuen, größeren Speicherbereich zu.

## 2.4 Vom Problem zum Programm

Bei Entwicklungsarbeiten muss häufig zuerst ermittelt werden, was überhaupt das Problem ist und was der Kunde tatsächlich haben möchte.

Ungeplantes „Losprogrammieren“ führt in aller Regel nicht zum erwarteten Erfolg.

Es gibt eine Vielzahl von Vorgehensmodellen, die den Entwicklern eine Hilfestellung im Vorgehen zur Problemerkennung und -lösung geben.

Innerhalb dieser Vorgehensmodelle kommen Entwicklungsmodelle zum Einsatz, die helfen, das Problem (meist graphisch) herauszuarbeiten und ebenso die Lösung zu gestalten. Die Problemerkennung und Lösungsgestaltung ist ein höchst kreativer Prozess. Ein Schritt-für-Schritt-Rezept kann es daher nicht geben. Allerdings existiert eine Vielzahl von Hilfsmitteln, die helfen typische Fehler zu vermeiden und die eine Kommunikation mit dem Kunden und anderen Entwicklern unterstützen sowie in der Strukturierung und Beförderung der Kreativität helfen.

### Intuitives Programmieren

- ohne Planung
- Funktionsumfang wird immer wieder erweitert
- Fehleranfälligkeit
- Schlecht zu warten
- Schlecht zu erweitern
- Spaghetticode



### Strukturiertes Vorgehen mit Phasen:

- Analyse (Analysis)
- Entwurf (Design)
- Implementierung (Implementation)

= Grundlegende Phasen der Entwicklung

### Grundsätzliches Vorgehen:

- Problem erfassen (Conceptualization / Requirements)
- Problem strukturieren (Structure)
  - Analyse der Beteiligten
  - Erkennen von Beziehungen, Kommunikation
- Problem abstrahieren (Abstraction)
  - Unbenötigte Teile weglassen
  - Modell vereinfachen
  - Aufgaben zuordnen
- Problem verfeinern (Refinement)
  - Benennung Akteure, Attribute
  - Restrukturierung (vereinigen, aufteilen)
  - Schnittstellen, Sichtbarkeiten

Die Vorgehen und die Entwicklungsphasen sind meist nicht zeitlich linear zueinander.



**Grundlegende Hilfsmittel zur Problemerkfassung und Gestaltung der Lösung:**

- Grafische Modellierung zur Unterstützung von Analyse, Entwurf und Implementierung sowie der Umsetzung der oben genannten grundsätzlichen Vorgehen.
- Als Hilfsmittel zur Darstellung von Problem und Problemlösung(sidee) sind beispielsweise UML Diagramme stark verbreitet.
- Häufig verwendete Hilfsmittel zur Beschreibung von Datenstrukturen, einer Programmiersprache bzw. der Bedienung eines Systems: BNF, EBNF, Syntaxdiagramme.

## 2.4.1 Modellierung mit UML

- UML (Unified Modeling Language)
  - Zusammenschluss der Methoden von James Rumbaugh, Grady Booch, Ivar Jacobson
  - Oktober 1995 (Rational Software Corporation)
  - IBM, HP
  - Januar 1997: UML Version 1.0
  - 2005 UML2
  - Standardisiert von der Object Management Group (OMG) [<http://www.uml.org>]
- UML ist eine „general-purpose“ graphische Modellierungssprache
  - zur Spezifikation,
  - zur Visualisierung,
  - zur Konstruktion und
  - zur Dokumentationder Artefakte eines Software Systems.
- UML umfasst eine Menge von graphischen Elementen, die nach bestimmten Regeln in verschiedenen Diagrammen (für die verschiedenen Sichten auf das System) kombiniert werden.
- UML umfasst viele verschiedene Diagrammarten zur Modellierung aus verschiedenen Sichten.
  - statische Sicht (Struktur), z.B. Klassendiagramme
  - dynamische Sicht (Verhalten), z.B. Sequenzdiagramme
- UML ist keine Programmiersprache  
stattdessen: Unabhängigkeit von Programmiersprachen
- Für UML gibt es eine Reihe von UML-Werkzeugen, die die grafische Modellierung unterstützen: Sie helfen
  - beim Zeichnen und Ändern von UML-Diagrammen,
  - bei der Erzeugung von Code aus UML-Diagrammen (eingeschränkt),
  - bei der Erzeugung von UML-Diagrammen aus Code (eingeschränkt).Beispiele für bekannte UML-Werkzeuge:  
ArgoUML, OMONDO, Together, Rational Rose (IBM).

**Bemerkung:**

Teile von Klassendiagrammen (v.a. Klassen-Darstellungen) haben wir schon in UML weiter vorn gesehen. Weitere Bausteine von UML-Klassendiagrammen und UML Sequenzdiagrammen werden wir im Weiteren ebenfalls begleitend kennenlernen.

## 2.4.2 Modellierung mit EBNF und Syntaxdiagrammen

- Zwei wichtige Beschreibungsformalismen für Datenstrukturen
  - BNF, EBNF (Extended Backus-Naur-Form)
  - Syntaxdiagramme
- Diese Beschreibungsmittel werden häufig verwendet zur Beschreibung der Syntax von Programmiersprachen oder auch allgemein zur Beschreibung der Bedienung von Systemen.

### Beispiel:

#### Deklaration von Java Attributen (vereinfacht) in EBNF:

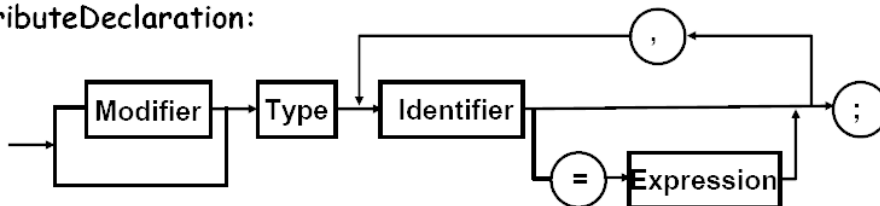
AttributeDeclaration ::= [Modifier] Type Identifier  
                                   ["=" Expression]  
                                   { "," Identifier  
                                   ["=" Expression] } ";"

Nicht-Terminal  
 Terminal

Modifier ::= "public" | "private"

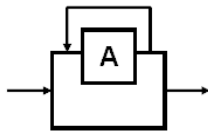

#### Deklaration von Java Attributen (vereinfacht) als Syntaxdiagramm:

AttributeDeclaration:



#### Notationselemente und Gegenüberstellung von EBNF und Syntaxdiagrammen:

EBNF:		Syntaxdiagramm:
A	Nicht-terminales Symbol	<div style="border: 1px solid black; padding: 2px; display: inline-block;">A</div>
"X" oder: 'X'	Terminales Symbol	<div style="border: 1px solid black; border-radius: 50%; padding: 2px; display: inline-block;">X</div> <div style="border: 1px solid black; border-radius: 15px; padding: 2px; display: inline-block;">X</div>
A B	Aneinanderreihung	<div style="display: inline-block; vertical-align: middle;"> <div style="border: 1px solid black; padding: 2px; display: inline-block;">A</div> <div style="display: inline-block; vertical-align: middle; font-size: 1.2em;">→</div> <div style="border: 1px solid black; padding: 2px; display: inline-block;">B</div> </div>
A   B	Alternative	<div style="display: inline-block; vertical-align: middle;"> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-bottom: 5px;">A</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-bottom: 5px;">B</div> </div>
[A]	Option	<div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 10px;">A</div>

{A}	beliebig häufige Wiederholung (inkl. Null)	
A ::=	zu definierendes nicht-terminals Symbol (Platzhalter)	A: Syntaxdiagramm  Name über dem Syntaxdiagramm
()	Gruppierung	

Die Überlagerung von Bild und Text in Zeile „A::=“ (rechte Spalte) soll ausdrücken, dass hier irgendein beliebiges Syntaxdiagramm stehen kann.

## 2.4.3 Muster in der Software-Entwicklung

Mit dem schon aus Kapitel 1 bekannten *Singleton* hatten wir eine erste Bekanntschaft mit Entwurfsmustern.

Ein Muster ist eine wiederverwendbare Lösung für häufig wiederkehrende Probleme bzw. eine Vorlage, wie ein bestimmtes Problem gelöst werden kann.

Die Idee zur Anwendung von Mustern ist in anderen Ingenieurdisziplinen (z.B. Architektur, Elektrotechnik) weit verbreitet.

In den 80iger/90iger Jahren wurde die Idee der systematischen Muster in der Software-Entwicklung „entdeckt“.

Muster gibt es in vielen Teilgebieten in der Softwareentwicklung.

Die ersten expliziten und bekanntesten Muster sind in der Kategorie der Entwurfsmuster (engl. Design Pattern) zu finden.

Definition: [GHJV94]

“A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in (object-oriented) systems.

The solution is a general arrangement of objects and classes that solves the problem. The solution is customized and implemented to solve the problem in a particular context.”

Das vermutlich bekannteste erste Buch zu Entwurfsmustern stammt von vier Personen, die kurz oft als *Gang of Four* (GoF) bezeichnet werden:

[GHJV94] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns*. Addison-Wesley, 1994

### Vorteile von Entwurfsmustern:

- Vereinfachung: Eine korrekte Nutzung hilft für eine Gestaltung guter und wartbarer Software.
- Erhöhung der Software-Qualität durch Wiederverwendung bewährter Lösungen.
- Vielfache Anwendbarkeit: Entwurfsmuster können untereinander kombiniert werden und in verschiedenen Anpassungsvarianten eingesetzt werden.

- Effizientere Kommunikation im Entwicklerteam: Die Menge der bekannten Muster bildet eine eigene Sprache. Verwendet eine Entwicklerin den Begriff „Singleton“ wissen bereits alle übrigen im Team, was sie genau damit meint. Die Entwicklerin muss nicht umständlich und aufwendig erklären, was gemeint ist.
- Implementierungsunterstützung: Entwurfsmuster geben Konventionen zur Benennung von Klassen und Methoden an und vereinfachen die Kommentierung (analog zur effizienteren Teamkommunikation).

**Nachteile und Fallen mit Mustern:**

- Aufwand: Die Muster müssen erst gelernt werden.
- Ressourcenproblem: Die Verwendung führt oft zu einem höheren Ressourcenbedarf als eine direkte Codierung. Die verbesserte Kapselung und Entkopplung wird meist mit mehr Methodenaufrufen und Klassen sowie komplizierteren Programmstrukturen bezahlt.
- Entwurfsverschlechterung: Werden Muster nicht passend angewendet, führen Sie schnell zu einer Verschlechterung des Entwurfs.
- Entwurfsmuster werden oft falsch als wiederverwendbare Einheiten verstanden. Entwurfsmuster sind kein Code und können stattdessen auf verschiedene Art und Weise implementiert werden (gültige Implementierungsvarianten!).
- Manche Entwurfsmuster sind trivial, um sie ernsthaft als Entwurfsmuster zu bezeichnen oder sie sind teils eigentlich nur Umgehungen für fehlende Unterstützung in der verwendeten Programmiersprache.

**Bemerkungen:**

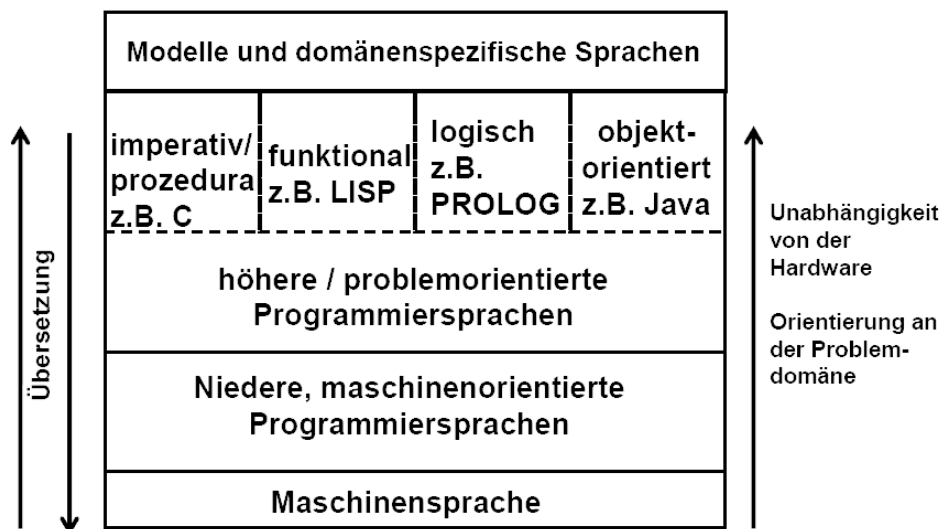
- Das schon kennengelernte Entwurfsmuster war hier nur verkürzt dargestellt. Jedes Muster hat (untrennbar) eine strukturierte, umfassende Form zur Dokumentation des Musters (→ Musterkatalog).
- Achtung: Die Anwendung der Entwurfsmuster sollte niemals das Ziel sein. D.h. die Vorstellung „je mehr Entwurfsmuster eingebaut sind, desto besser“ ist grundsätzlich falsch. Vielmehr sollte ausgehend von einem Entwurfsproblem geschaut werden, welche Muster zu dem Problem vorhanden sind. Gegebenenfalls hilft das eine oder andere für eine gute Lösung.
- Einige Entwurfsmuster werden wir im Weiteren - wie schon das Singleton-Entwurfsmuster - begleitend kennenlernen.

## 2.5 Programmiersprachenparadigmen

Ist das Problem und die Lösung modelliert, muss irgendwann der Schritt in die Umsetzung gemacht werden: die Implementierung.

Spätestens vor der Implementierung muss entschieden werden, welche Sprache eingesetzt werden soll. (Bemerkung: In der Regel geschieht das schon viel früher.)

Eine erste grundsätzliche Entscheidung ist, welches Programmierparadigma in der Implementierung verfolgt werden soll.



Programmierparadigmen hatten wir zu Beginn schon kurz angesprochen:

- Ein Programmiersprachenparadigma liefert (und bestimmt) die Sicht auf die Programmausführung.
- Ein Programmiersprachenparadigma ist eine Basis zur Klassifikation von Sprachen.
- Verschiedene Programmiersprachen unterstützen verschiedene Paradigmen (mehr oder weniger gut).
- Eine Programmiersprache kann mehrere Paradigmen unterstützen (z.B. C++ als hybride Programmiersprache).

**Zur Einschätzung der Häufigkeit der eingesetzten Paradigmen:**

Category	Ratings Apr 2011	Delta Apr 2010
Object-Oriented Languages	58.8%	+4.8%
Procedural Languages	36.4%	-5.2%
Functional Languages	3.7%	+0.6%
Logical Languages	1.2%	-0.3%

Aus: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

Bemerkung:

Die Paradigmen spielen nicht erst in der Programmierung eine große Rolle, sondern auch schon zuvor in der Sichtweise und Herangehensweise im Rahmen der Modellierung.