

## Java-Einführung

**Autor:** Dipl.-Math.  
E. Engelhardt

**Stand:** 16. Mai 2009



# Abkürzungen

- ◆ API Application Programming Interface
- ◆ AWT Abstract Window Toolkit
- ◆ GUI Gaphical User Interface
- ◆ IDE Integrated Development Environment
- ◆ IO Input Output (Ein- Ausgabeströme)
- ◆ JDK Java Development Kit
- ◆ JFC Java Foundation Classes
- ◆ JRE Java Runtime Environment
- ◆ JVM Java Virtual Machine
- ◆ SDK Software Development Kit

# Inhalt

## 1 Einleitung

- 1.1 Geschichte von Java
- 1.2 Internet und Java
- 1.3 Java-Entwicklungsstufen

## 2 Aufbau von Java

- 2.1 Architektur des sdk 1.3
- 2.2 Architektur des sdk 1.4
- 2.3 Architektur des sdk 1.5
- 2.4 Architektur des Java 6
- 2.5 Komponenten des Java 6

## 3 Java-Bedeutung

- 3.1 Java und JavaScript
- 3.2 Was ist Java?
- 3.3 Java als Programmiersprache
- 3.4 Java als Plattform
- 3.5 Was kann Java?

## 4 Java-Programmierung

- 4.1 Das Java-API
- 4.2 Was soll Java?
- 4.3 Ein C-Programm ausführen
- 4.4 Ein Java-Programm ausführen
- 4.5 „Hallo Welt“-Applikation
- 4.6 „Hallo Welt“-Applet
- 4.7 Allgemeine Form einer Klasse
- 4.8 Zugriffsberechtigungen

## 5 Java-Sprachbeschreibung

- 5.1 Erweiterte Backus-Naur-Syntax
- 5.2 Grundsätzlicher Aufbau
- 5.3 Elementare Datentypen
- 5.4 Vereinbarungen
- 5.5 Arithmetische Operatoren

# Inhalt

- 5.6 Zuweisungen, Bedingungsoperator
- 5.7 Vergleichs- und logische Operatoren
- 5.8 Ausdruck, Anweisung

## 6 Kontrollstrukturen, Ablaufsteuerung

- 6.1 **if**-Verzweigung
- 6.2 **for**-Schleife
- 6.3 **while**-Schleife
- 6.4 **switch**-Mehrfachverzweigung
- 6.5 Schlüsselwörter **continue** und **break**
- 6.6 goto, return und exit

## 1.1 Geschichte von Java

- ◆ Team um James Gosling (Autor des UNIX-emacs)
- ◆ Sprache zur Entwicklung von Software für Konsumgüter
- ◆ Dafür sind „Hochsprachen“ (C/C++, usw. ungeeignet)
- ◆ Deshalb seit 1990 Entwicklung von Java
- ◆ Angelehnt an C/C++
- ◆ James Gosling wählte den Namen „Oak“ (Eiche)
- ◆ „Oak“ gab es schon, deshalb „Java“
- ◆ „JavaSoft“ ist Tochterfirma von Sun Microsystems

## 1.2 Internet und Java

- ◆ Anfangs nur Daten- und Dokumentenaustausch
- ◆ 1989 Tim Berners-Lee (Physiker am CERN) Konzept des WWW
- ◆ 1992 wurde WWW international bekannt
- ◆ erste Web-Browser („Mosaic“) durch NCSA (National Center for Supercomputing Applications)
- ◆ WebRunner: Web-Browser des Java-Teams (vollständig in Java geschrieben)
- ◆ Umbenennung in HotJava durch Urheberrechte
- ◆ HotJava demonstrierte Bedeutung von Java für Internet
- ◆ Mai 1995 SunWorld-Konferenz (San Francisco)  
Publizierung der Java-Technologie
- ◆ Netscape soll Java unterstützen

## 1.3 Java-Entwicklungsstufen (1/4)

- ◆ JDK 1.0 (Java Development Kit 1.0) Jahr 1996
  - Grundlegende Objekte (Objekte, Strings, Threads, Ein- und Ausgabe, Datenstrukturen, Systemeigenschaften)
  - Grafikprogrammierung (awt: **a**bstract **w**indows **t**oolkit)
  - Netzwerkbetrieb (URLs, TCP- und UDP-Sockets)
- ◆ JDK 1.1 Jahr 1997
  - Vereinfachung von GUI-Programmierung (Eventhandling)
  - Internationalisierung
  - Sicherheit (verschiedene Stufen realisierbar)
  - JavaBeans, JAR-Format, Datenbankabbinding
  - RMI (Remote Method Invocation), Reflections)
  - JNI (Java Native Interface)
  - mathematische Hilfsklassen (java.math)

## Java-Entwicklungsstufen (2/4)

- ◆ JDK 1.2.x (Java 2) Jahr 1998
  - Sicherheitskonzept (Zertifizierung)
  - JFC (Java Foundation Classes)
    - Java-swing, Java 2D, Drag and Drop
    - leichtere Interaktion von Java-Programmen mit Außenwelt
  - Collections-API (Java Collections Applications Programming Interface) meist Daten-Container
  - IDL (Java Interface Definition Language) für CORBA (Common Object Request Broker Architecture)
  - weitere Erweiterungen (Audio, Versionsidentifikation, usw.)

## Java-Entwicklungsstufen (3/4)

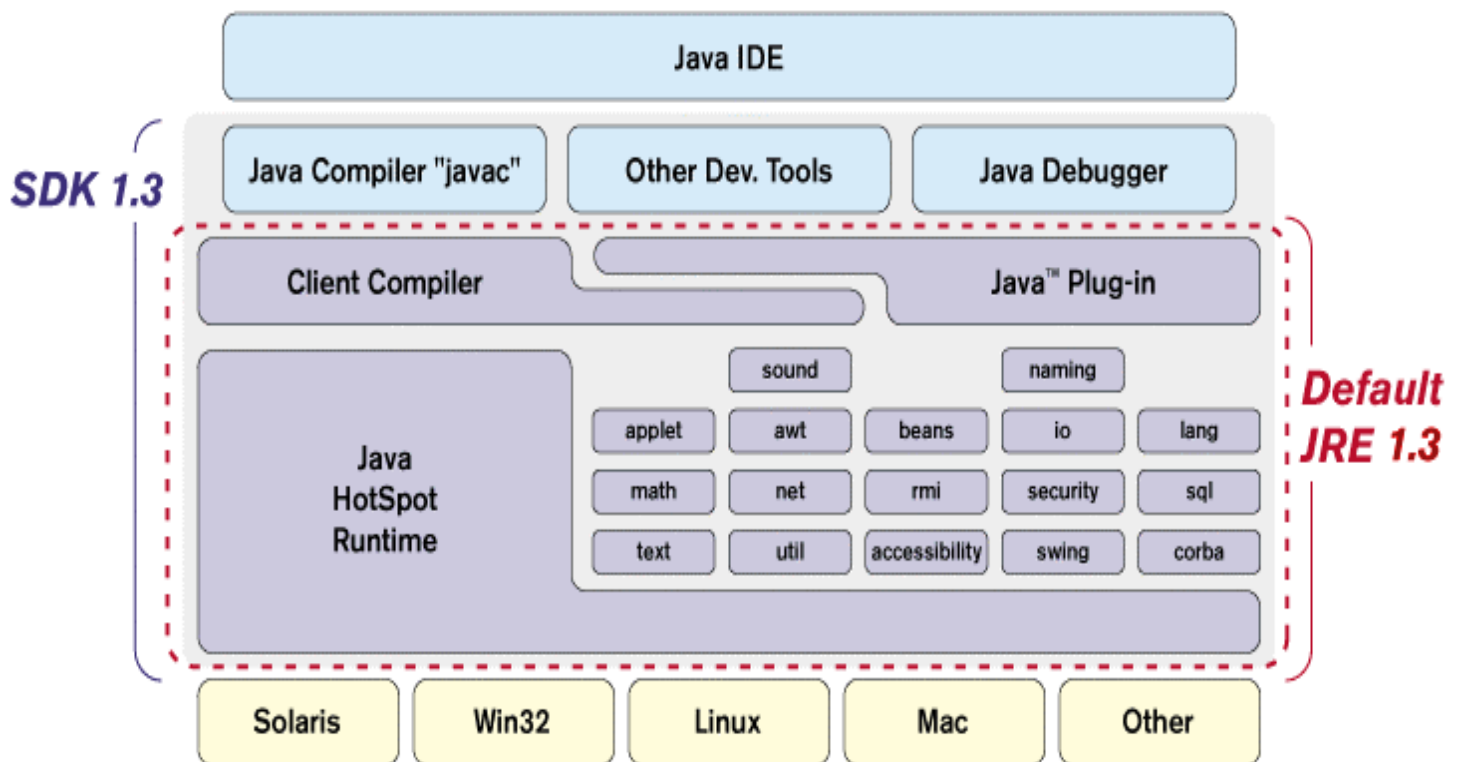
- ◆ JDK 1.3.x (Java 2) „Kestrel“ Jahr 2000
  - Einführung der HotSpot-Technologie
  - Java-Plug-In steht unter Windows über die Systemsteuerung zur Verfügung
- ◆ JDK 1.4.x Jahr 2002
  - Keine klassische JVM (ohne Hotspot-Optimierung) mehr
  - Unterstützung von Unicode der Version 3.0
  - weitere neue I/O APIs
  - Java-Plug-In 1.4 u.a. mit Multiversionssupport
  - Securityfeatures JCE, JSSE und JAAS
  - Web-Start-Technologie
  - XML

## Java-Entwicklungsstufen (4/4)

- ◆ Version 5.0 der Java 2 Standard Edition („Tiger“)
- ◆ Die Java-Programmierung wird durch folgende Erweiterungen der Edition J2SE 1.5 erleichtert:
  - Autoboxing/Autounboxing (zwischen primitiven/elementaren Datentypen und Wrapperklassen)
  - Erweiterung der for-Schleife („foreach“ – kein Schlüsselwort)
  - Variable Parameterlisten
  - Statische Imports („static import“)
  - Aufzählungstypen („enum“)
  - Ausgabeformatierung („out.format(...“)
  - Generische/typisierte Klassen und Connections

## 2.1 Architektur des sdk 1.3

Abbildungen der folgenden Seiten aus „java.sun.com“

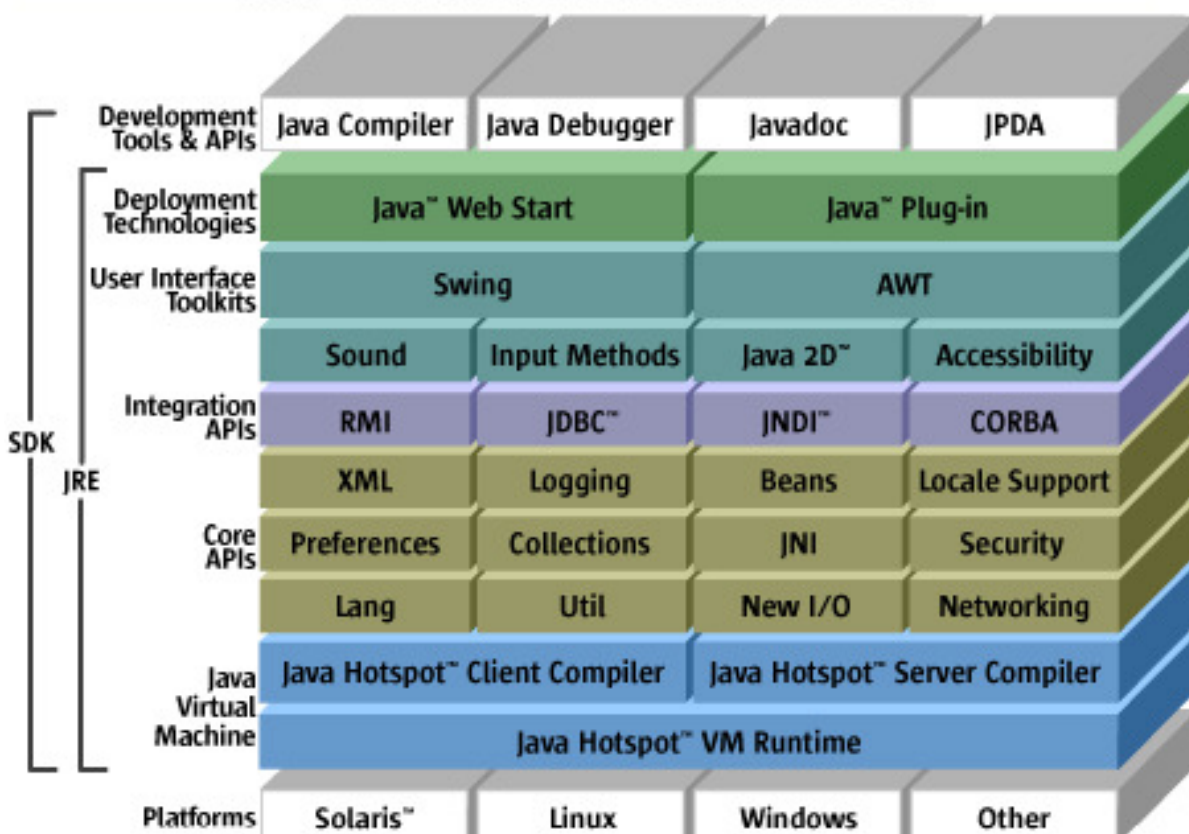


Folie 13 von 70

Java-Einführung

## 2.2 Architektur des sdk1.4

### Java™ 2 Platform, Standard Edition v 1.4

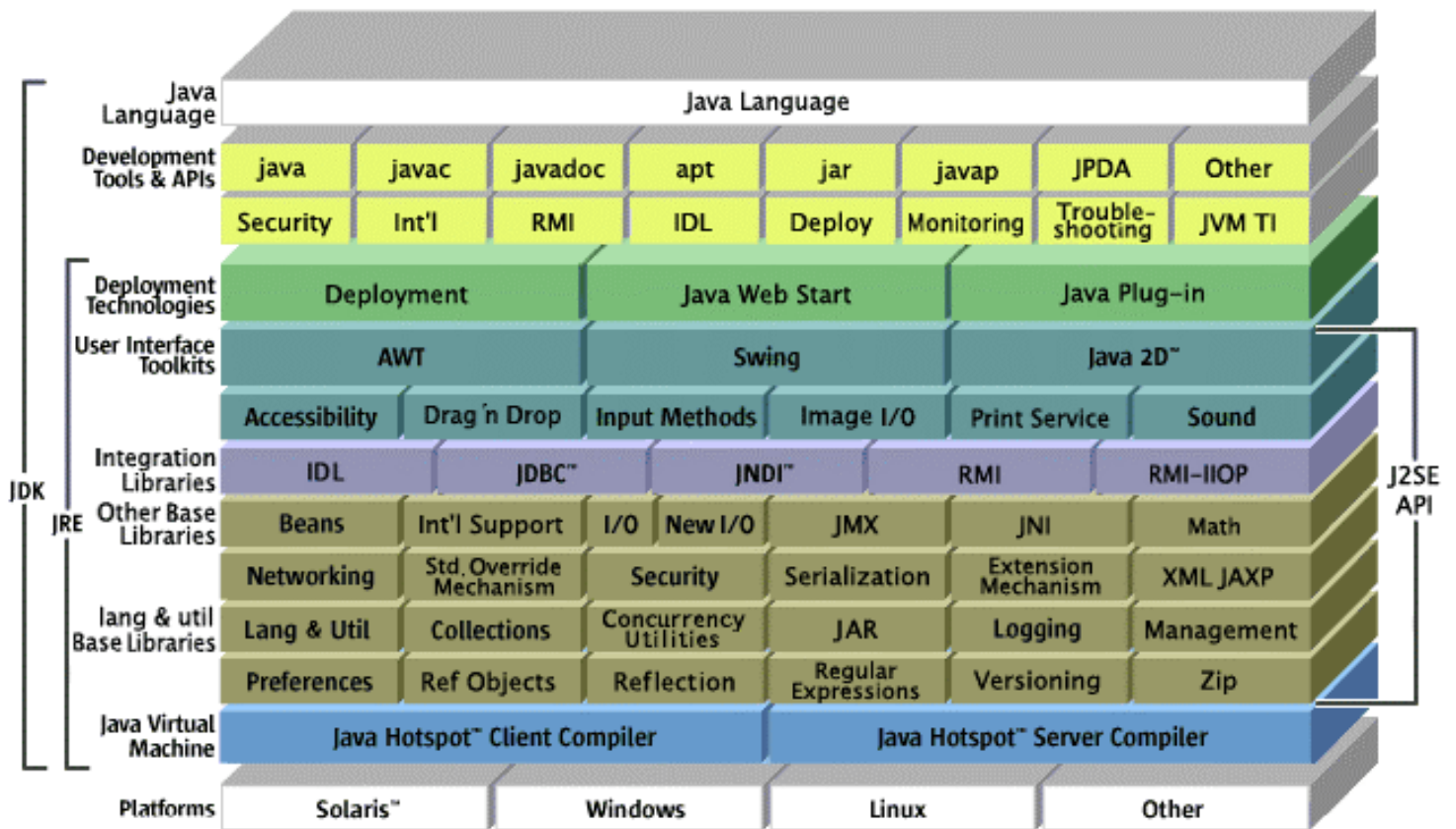


Folie 14 von 70

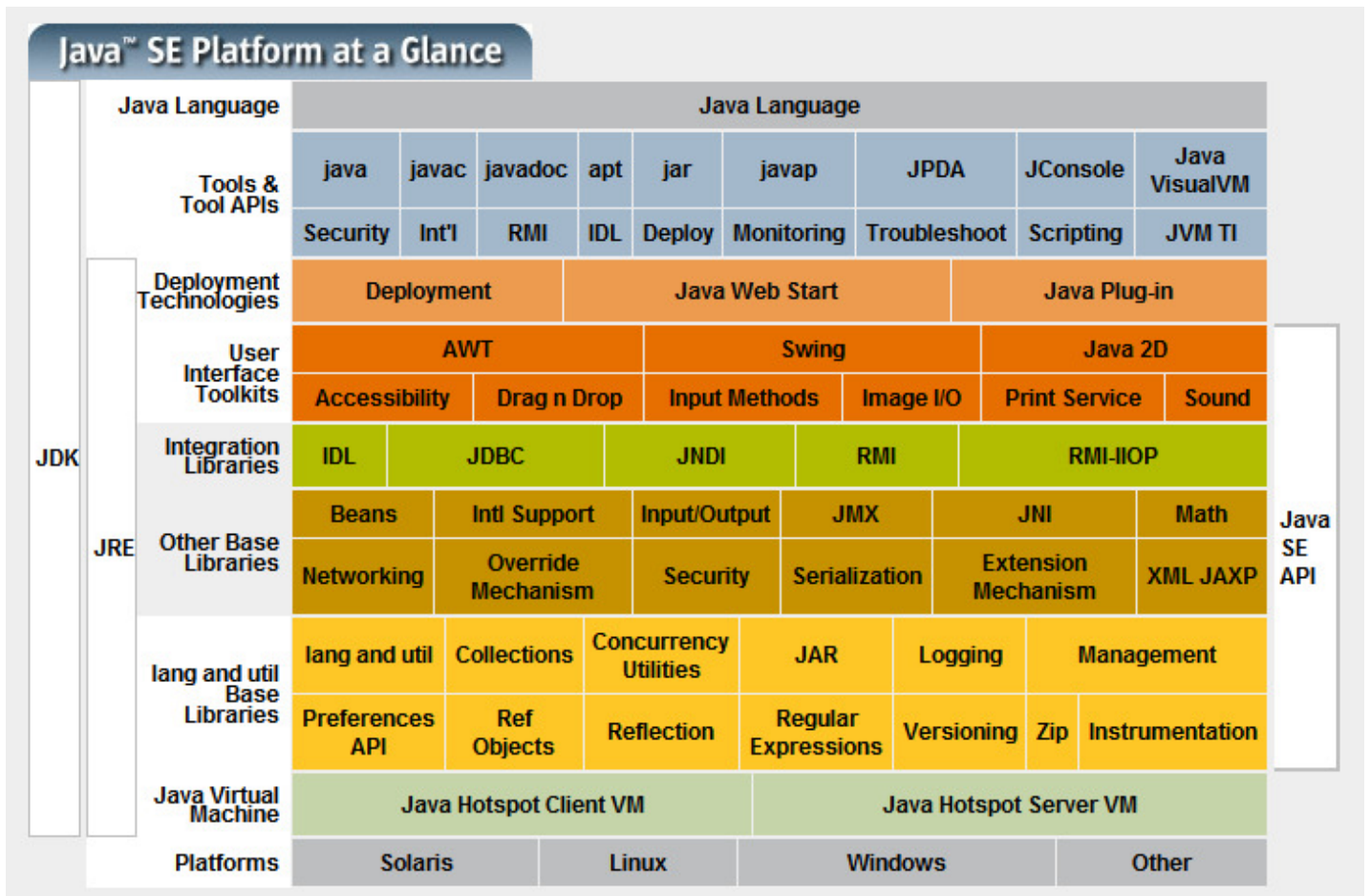
Java-Einführung

## 2.3 Architektur des jdk 1.5 „J2SE 5.0“

### Java™ 2 Platform Standard Edition 5.0



## 2.4 Architektur des Java 6



## 2.5 Pakete des Java 6 Standardpakete

- ◆ java.applet Applets
- ◆ java.awt Abstract Windowing Toolkit
- ◆ java.beans Java Bean
- ◆ java.io Datei-I/O
- ◆ java.lang elementare Sprachunterstützung
- ◆ java.lang.ref Referenz-Objekte
- ◆ java.lang.reflect Reflection-API
- ◆ java.math mathematische Funktionen
- ◆ java.net Netzwerkunterstützung
- ◆ java.nio new I/O (seit Java 5.0)
- ◆ java.rmi verteilte Anwendungen (RMI)
- ◆ java.security Security-Dienste
- ◆ java.sql Datenbankzugriff (JDBC)
- ◆ java.text Internationalisierung
- ◆ java.util Utilities, Collections

## Standarderweiterungen

- ◆ javax.accessibility E/A-Geräte (Braille-Zeile)
- ◆ javax.crypto Kryptografische Erweiterungen
- ◆ javax.imageio Lesen und Schreiben von Bilddateien
- ◆ javax.naming Zugriff auf Namens-Services
- ◆ javax.print Unterstützung zum Drucken
- ◆ javax.security.auth Authentifizierung und Autorisierung
- ◆ javax.sound Sound-API
- ◆ javax.swing SWING-Toolkit
- ◆ javax.xml XML-Dateien

### Weitere Java-Erweiterungen

- ◆ Java3D 3D-Erweiterung (OpenGL oder DirectX)
- ◆ JAI Java Advanced Imaging
- ◆ JMF Java Media Framework

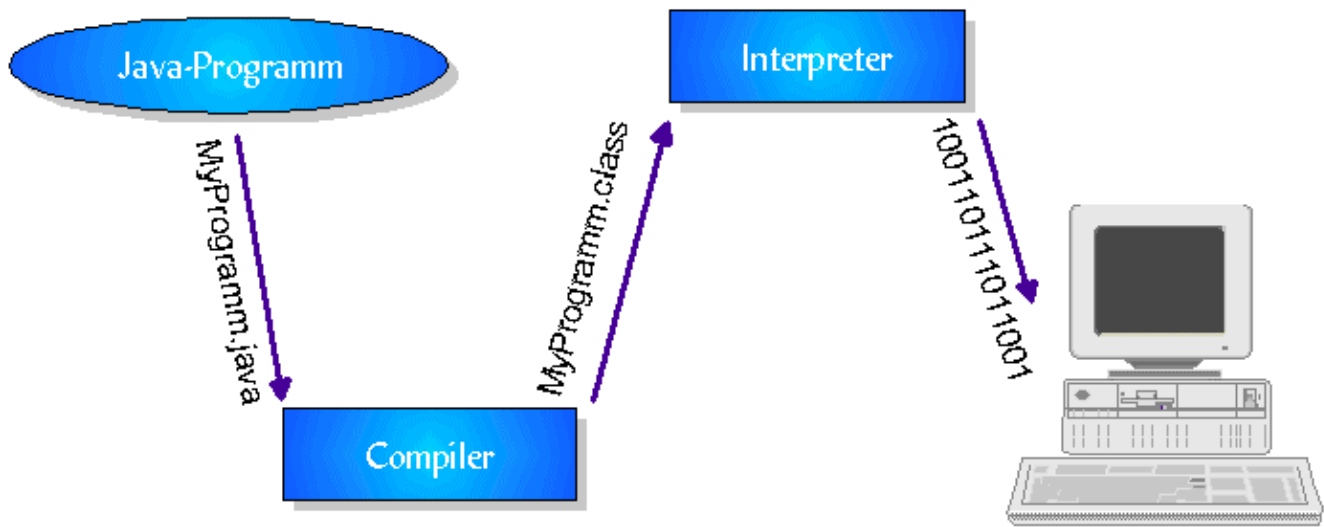
## 3.1 Java und JavaScript

- ◆ Entwicklung der Firma „Netscape Corporation“
- ◆ JavaScript ist Makroprogrammiersprache
- ◆ JavaScript wird nicht übersetzt und ist im Quelltext in der HTML- Seite eingefügt (<SCRIPT> ... </SCRIPT>)
- ◆ Objektbasierend (nicht streng objektorientiert)
- ◆ Im Vergleich der Möglichkeiten eine Untermenge von Java

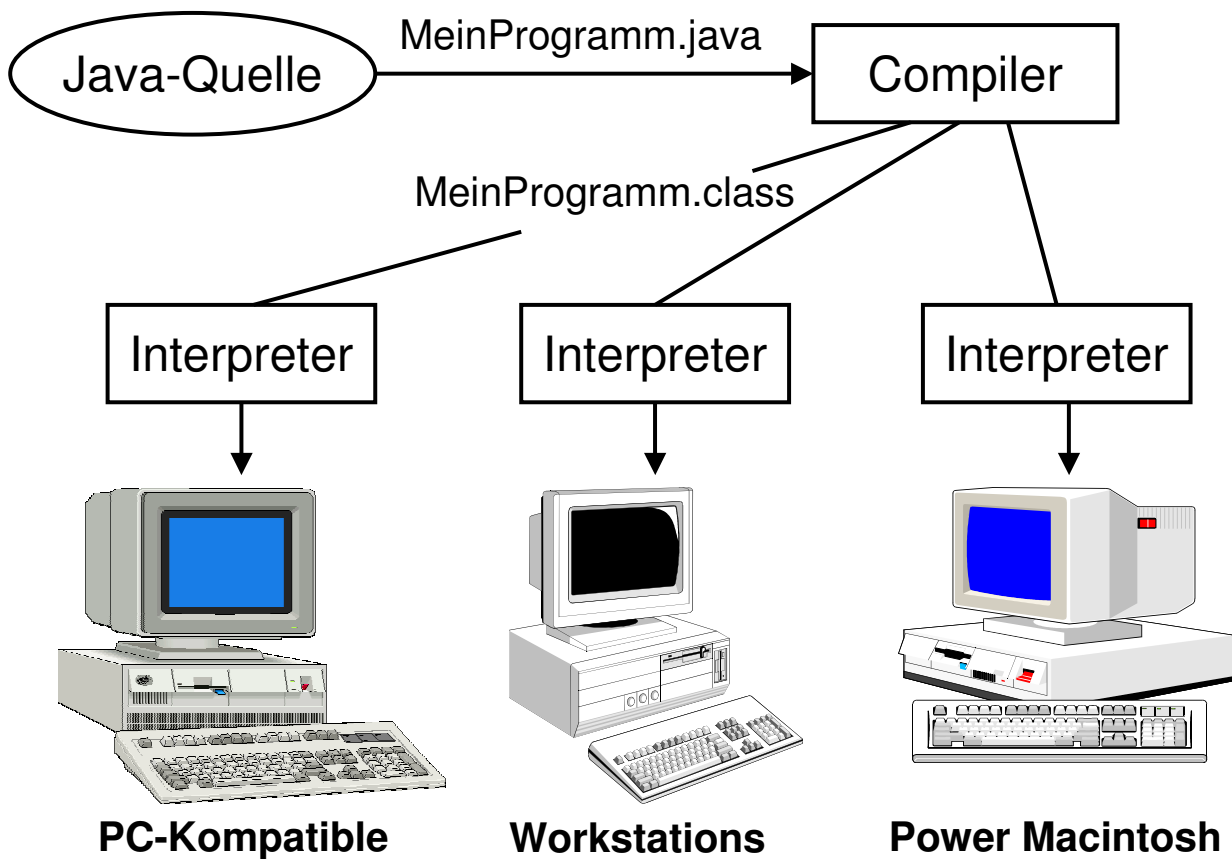
## 3.2 Was ist Java?

- ◆ Java ist zweierlei: eine Programmiersprache und eine Plattform
- ◆ Eigenschaften von Java als Sprache:
  - einfach
  - architekturneutral
  - objektorientiert
  - portabel
  - verteilt
  - Interpretersprache
  - multithreaded
  - hohe Performance
  - robust
  - dynamisch
  - sicher

### 3.3 Java als Programmiersprache



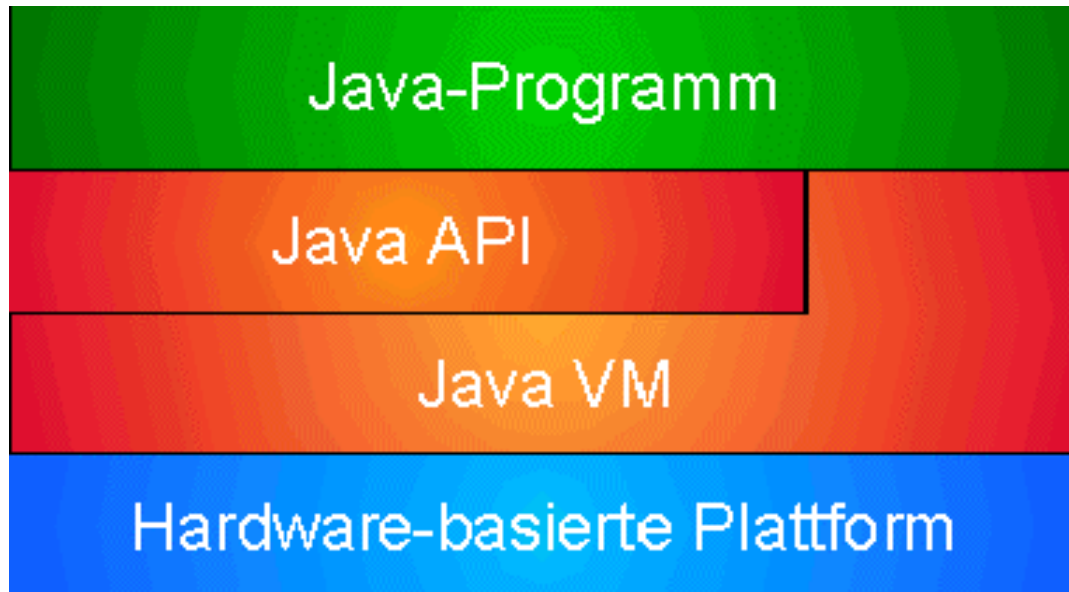
„Write once, run everywhere“



## 3.4 Java als Plattform

Die Java-Plattform hat zwei Komponenten:

- ◆ Java Virtual Machine (JVM)
- ◆ Java Application Programming Interface (Java API)



## 3.5 Was kann Java?

Java Programme werden unterschieden in:

- ◆ Applications = eigenständige Programme
- ◆ Applets = Programme, die in einem anderen Programm (Webbrowser) ablaufen
- ◆ Server, z.B. Webserver, Mailserver, ...
- ◆ Servlets, ähnlich einem Applet, eine Art Server-Erweiterung
- ◆ JSP Java Server Pages (entspr. ASP)

## 4.1 Das Java API (1/3)

Jede vollständige Java-Implementation enthält das sogenannte Core-API:

- ◆ Essentials: Objekte, Zahlen, Zeichenketten, Ein- und Ausgabe, Datum/Zeit, ...
- ◆ Applets: alle Konventionen für Java Applets
- ◆ Networking: URLs, TCP/UDP, IP, Sockets
- ◆ Internationalization: Unterstützung für mehrsprachige Programme

## Das Java API (2/3)

- ◆ **Security:**  
Elektronische Unterschriften, private/öffentliche Schlüssel, Zugriffskontrolle, Zertifikate
- ◆ **Software components:**  
JavaBeans, die sich auch in existierende Komponentenarchitekturen einklinken können, wie z.B. OpenDoc, Netscapes Live Connect, OLE/COM/Active-X
- ◆ **Object serialization:**  
Kommunikation via RMI
- ◆ **Java Database Connectivity (JDBC):**  
Einheitlicher Zugriff auf ein breites Spektrum relationaler Datenbanken

Zusätzlich zum Core-API gibt es noch Erweiterungen, zum Beispiel:

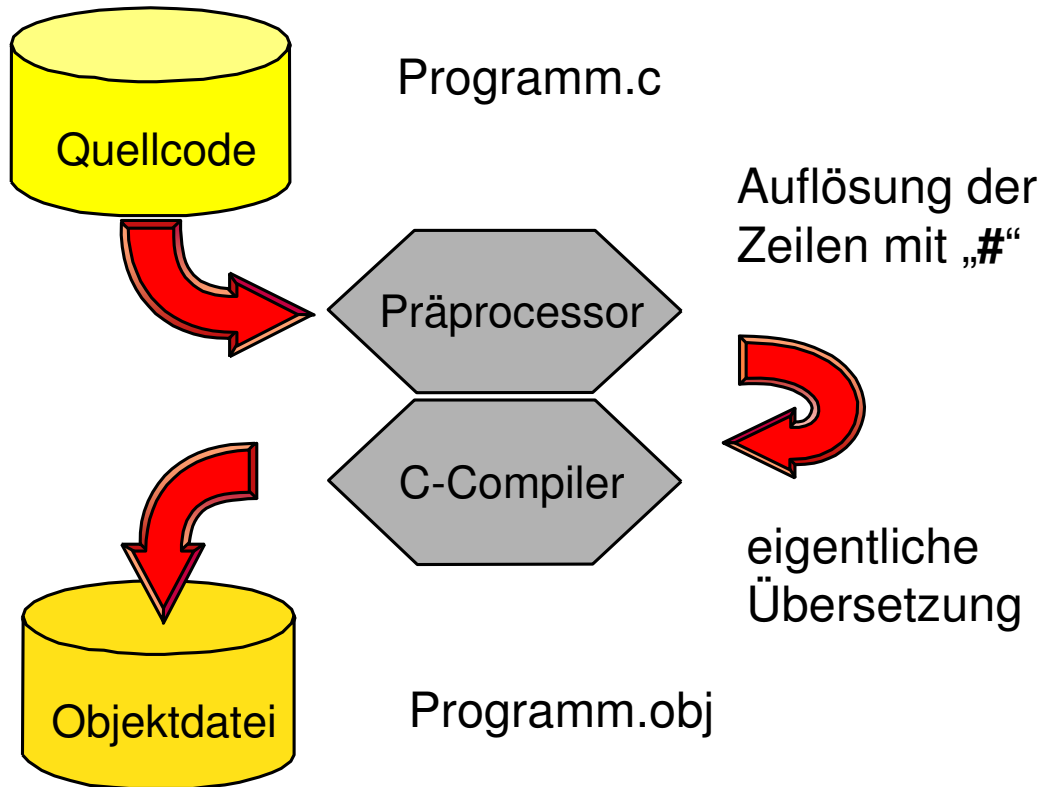
- ◆ Java-3D
- ◆ JAI (Java Advanced Imaging)
- ◆ JMF (Java Media Framework)
- ◆ Telephonie
- ◆ Server
- ◆ Sprachausgabe
- ◆ Animationen
- ◆ ...

## 4.2 Was soll Java?

Java kann bei Folgendem helfen:

- ◆ einen schnellen Einstieg finden
- ◆ weniger Code schreiben zu müssen
- ◆ leichter besseren Code schreiben zu können
- ◆ Programme schneller entwickeln zu können
- ◆ Plattformunabhängigkeit durch „100% pure Java“
- ◆ „Write once, run everywhere“
- ◆ Anwendungen leichter verteilen
- ◆ ...

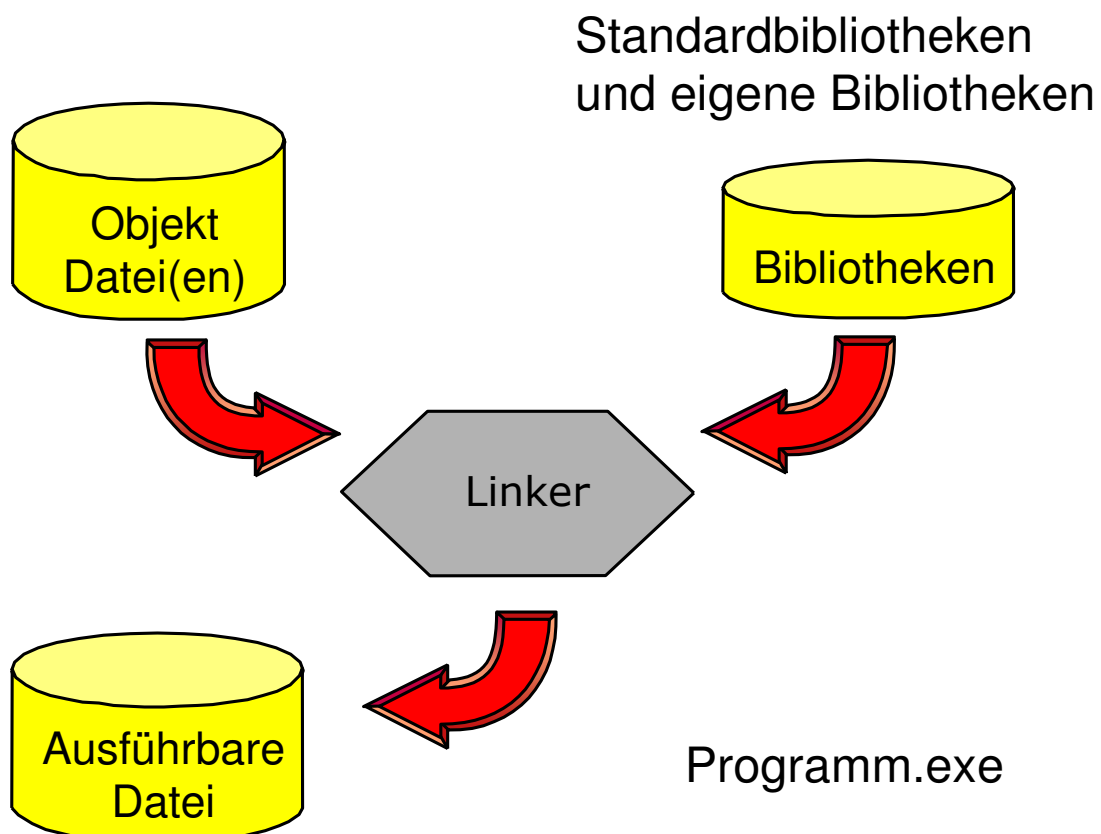
## 4.3 Ein C-Programm ausführen (1/3)



Folie 29 von 70

Java-Einführung


## Ein C-Programm ausführen (2/3)



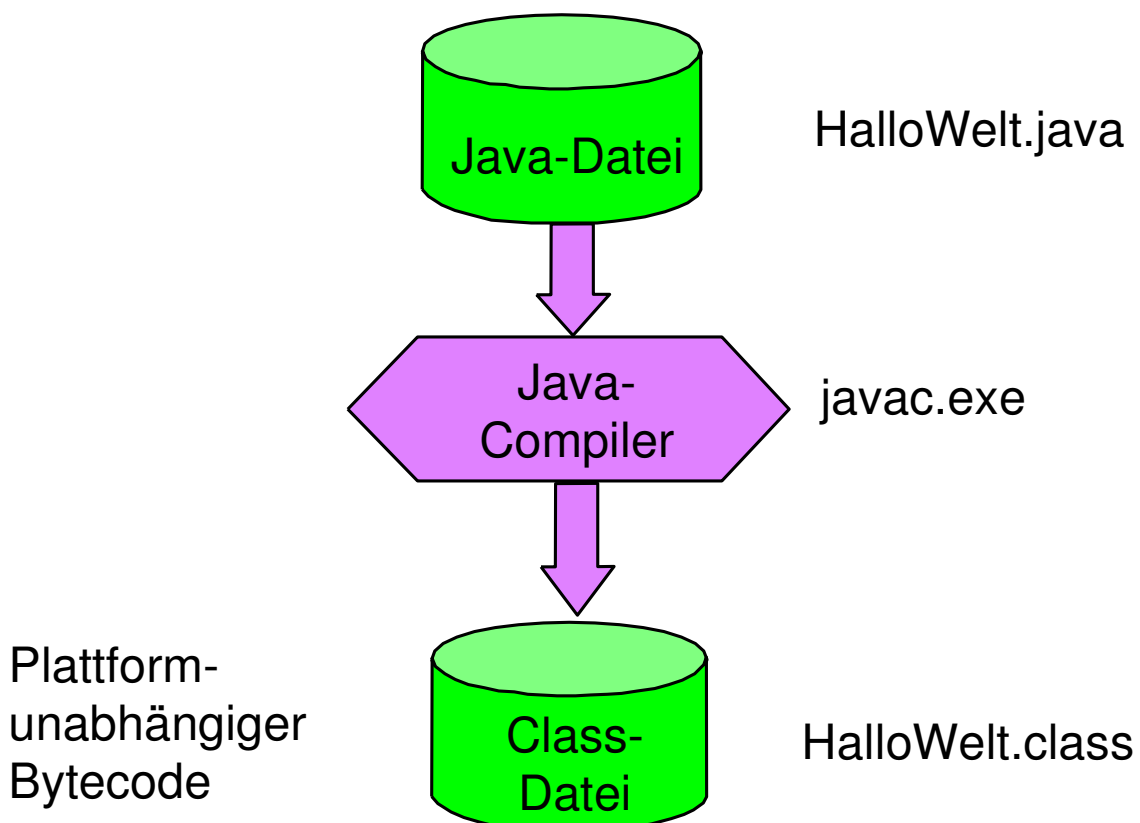
Folie 30 von 70

Java-Einführung

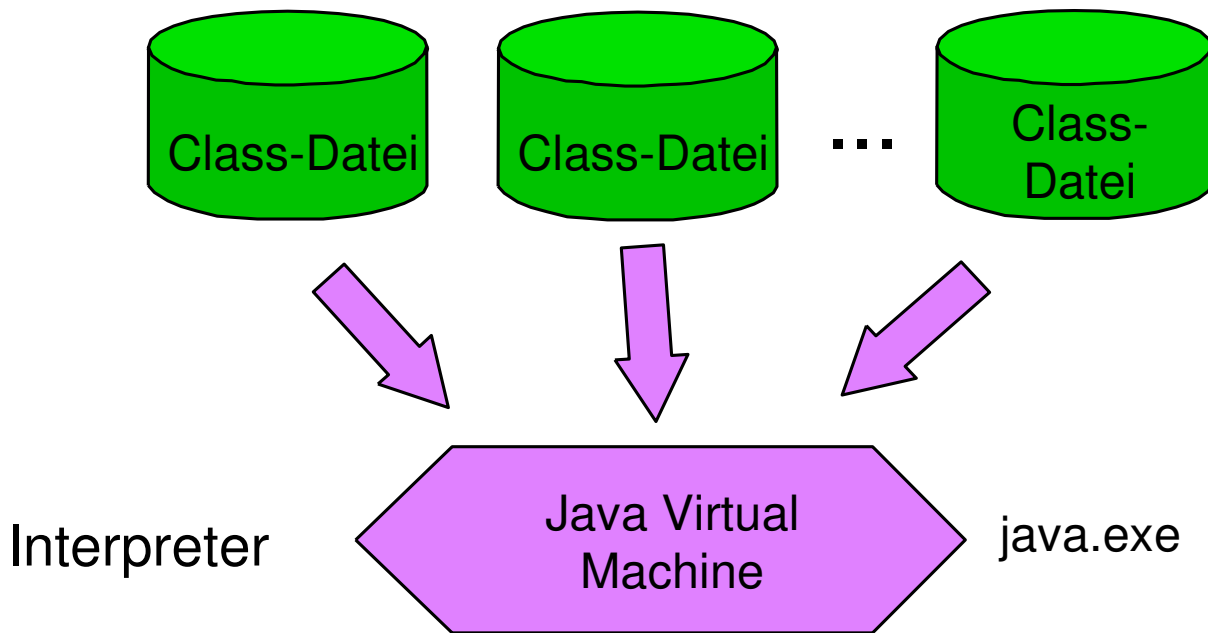
## Ein C-Programm ausführen (3/3)

- ◆ C-Quellcode muss auf jeder Plattform mit dem dort verfügbaren Compiler neu übersetzt werden
  - ◆ Auch die Bibliotheken müssen für die jeweilige Plattform (Prozessor und Betriebssystem) erstellt sein.
  - ◆ Der Linker erstellt ein ausführbares Programm für die entsprechende Plattform
- 
- ◆ Nur auf binärkompatiblen Plattformen kann das Programm ausgeführt werden.
  - ◆ Mit ANSI-C kann zumindest Quellcode-Kompatibilität erreicht werden.

## 4.4 Ein Java-Programm ausführen (1/3)



## Ein Java-Programm ausführen (2/3)



## Ein Java-Programm ausführen (2/3)

- ◆ Der Java-Quellcode wird durch den Java-Compiler in einen maschinencodenahen, plattformunabhängigen Bytecode übersetzt. Dieser Bytecode wird in der Datei gleichen Namens wie der Java-Quellcode (... .class) abgelegt.
- ◆ Der Java-Bytecode wird mittels JVM (Java Virtual Machine) auf der jeweiligen Plattform ausgeführt.



- ◆ Der Bytecode muss nur einmal erstellt werden. Er ist auf jeder Plattform mittels der dort vorhandenen JVM ausführbar.

# Applikationen versus Applets

## Applikationen

- ◆ grafisch oder textbasiert
- ◆ Zugriff auf alle Systemressourcen
- ◆ werden über die JVM vom Benutzer gestartet
- ◆ enthalten mindestens ein „main“
- ◆ Klassen werden nur von der Festplatte geladen

## Applets

- ◆ nur grafisch
- ◆ Sicherheitskonzept; Zugriff nur auf bestimmte Systemressourcen
- ◆ eingebettet in HTML-Seiten
- ◆ werden gestartet, wenn HTML-Seite aufgerufen wird
- ◆ Festplatte oder Netzwerk zum Laden der Klassen

## 4.4 „Hallo Welt“-Applikation

### HalloWelt.java

```
import java.io.*;

public class HalloWelt
{
    public static void main(String[] arguments)
    {
        System.out.println("\nHallo Welt!");
    }
}
```

# Applikation-Grundgerüst

- ◆ Eine Applikation (Anwendung) besteht aus mindestens einer public-Klasse, die Methode „main“ enthält
- ◆ Der Name der public-Klasse ist identisch mit dem Dateinamen (vor dem Punkt für Datei-Typ)
- ◆ Eine Applikation kann mehrere public-Klassen mit „main“ enthalten
- ◆ Eine Applikation hat nur dann eine Grafikoberfläche, wenn die public-Klasse entsprechend abgeleitet wurde („extends“) und wenn die Grafikpakete mittels „import“ eingebunden wurden
- ◆ Für Ausgaben mittels „System.out.println(...)“ muss nicht „import java.io.\*;“ geschrieben werden

## Java-Programmentwicklung ohne IDE (nur zur Veranschaulichung)

```
D:\MeinJava> set PATH=%PATH%;...\jdk1.5.0\bin  
bewirkt, dass „javac.exe“ gefunden wird!
```

```
D:\MeinJava> edit HalloWelt.java  
Editieren der Quelle mit beliebigem Texteditor
```

```
D:\MeinJava> javac HalloWelt.java  
Java-Bytecode wird erzeugt und in Datei „HalloWelt.class“  
abgelegt
```

```
D:\MeinJava> java HalloWelt  
Hallo Welt!
```

```
D:\MeinJava>
```

## 4.6 „Hallo Welt“-Applet (1/2)

### HalloWeltApplet.java

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

public class HalloWeltApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Hallo Welt!", 50, 100);
    }
}
```

## Hallo Welt Applet (2/2)

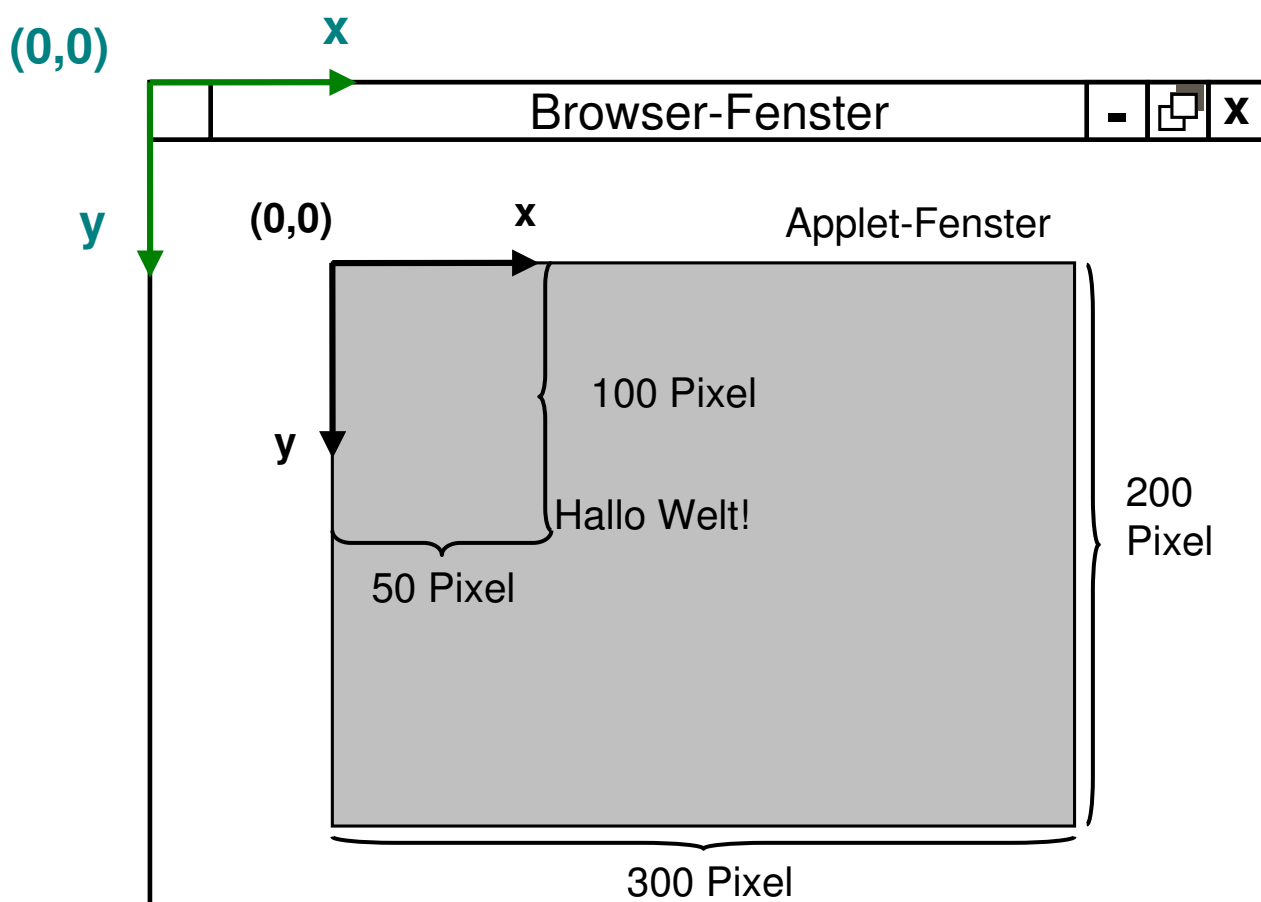
### HalloWeltApplet.html

```
<html>
<body>
    <applet code="HalloWeltApplet.class"
            width="300" height="200">
    </applet>
</body>
</html>
```

# Applet-Grundgerüst

- ◆ Ein Applet besteht aus mindestens einer public-Klasse ohne Methode „main“
- ◆ Name der public-Klasse ist identisch mit dem Dateinamen (vor dem Punkt für Datei-Typ)
- ◆ Da ein Applet Teil eines Fensters (Panel) einer übergeordneten grafikorientierten Anwendung (Applet-Viewer oder Browser) ist, ist es auch automatisch grafikorientiert (Applet läuft in „sandbox“).
- ◆ Zur Ausführung eines Applets muss immer eine entsprechende HTML-Datei vorhanden sein
- ◆ In der Methode „paint“ erfolgen die Grafikausgaben

## Koordinatensystem und Einheiten



## Hinweise zu Applets

- ◆ Die Lage des Applets im Browserfenster wird durch die HTML-Tags bestimmt (Table, Frame, ...)
- ◆ Die Größe des Applets steht im Applet-Tag (width und height); Angaben in Pixel
- ◆ Der Internet-Explorer stellt den Applet-Hintergrund standardmäßig grau dar!
- ◆ Bei Ausführung mittels Appletviewer wird nur das Applet-Fenster angezeigt
- ◆ Der zum jdk gehörige Appletviewer sollte das Applet auf jeden Fall fehlerfrei abarbeiten!
- ◆ Die verschiedenen Browser können unterschiedlich reagieren

## Java-Applet ohne IDE

D:\MeinJava> set PATH=%PATH%; ... \jdk1.3\bin  
bewirkt, dass „javac.exe“ gefunden wird!

D:\MeinJava> edit HalloWeltApplet.java  
Editieren der Quelle mit beliebigem Texteditor

D:\MeinJava> javac HalloWeltApplet.java  
Java-Bytecode wird erzeugt und in Datei  
„HalloWeltApplet.class“ abgelegt

D:\MeinJava> edit HalloWeltApplet.html  
HTML-Datei kann anderen Namen haben!

D:\MeinJava> appletviewer HalloWeltApplet.html  
oder HTML-Datei mit Browser öffnen

## 4.7 Allgemeine Form einer Klasse

```
public class KlassenName extends Superklasse
{
    protected typ variableName;
    // Variablen fangen mit Kleinbuchstaben an

    public typ methodenName (typ parameterName, ...)
    {
        // Anweisungen fangen mit
        // Kleinbuchstaben an
    }
    // ...
}
```

## 4.8 Zugriffsberechtigungen

- ◆ public
  - ◆ Jeder kann auf das Datum / die Methode zugreifen
- ◆ private
  - ◆ Nur die eigene Klasse kann auf das Datum / die Methode zugreifen.
  - ◆ Auch abgeleitete Klassen haben keinen Zugriff
- ◆ protected
  - ◆ Wie private, aber abgeleitete Klassen können zugreifen

## 5.1 Erweiterte Backus-Naur-Syntax (EBNF)

- ◆ `[ ... ]` was zwischen den eckigen Klammern steht kann, muss aber nicht stehen.
- ◆ `... | ...` es gilt eine der Alternativen (die vor oder die nach dem Senkrechtrich)
- ◆ `< ... >` was zwischen den spitzen Klammern steht ist ein Platzhalter für einen Bezeichner
- ◆ `,` Komma ist Trenner bei Aufzählungen
- ◆ `{ ... }` beliebige Anzahl des in Klammern stehenden Syntax-Elementes
- ◆ `{ ... }1` wie oben, mindestens einmal

## 5.2 Grundsätzlicher Aufbau (1/3)

- ◆ **Sammlung von Klassen, auch über mehrere Dateien (Module)**
  - mindestens eine Klasse, die genau so heißen muss wie die Datei (ohne Extension „.java“) und „public“ sein muss
  - mindestens eine Methode „main“ in einer Applikation
  - in „main“ wird Instanz der „Hauptklasse“ gebildet
- ◆ **Klassen bestehen aus:**
  - Attributen (Variable, Objekte)
  - Methoden (Funktionalität)
  - evtl. Konstruktoren bei Applikationen
  - Methoden „init“, „start“ und „stop“ bei Applets

## Grundsätzlicher Aufbau (2/3)

- ◆ Außerhalb von Klassen gibt es keine Methoden.
- ◆ Klassen sind die allgemeinen Beschreibungen. Im Programm wird mit Objekten gearbeitet
- ◆ Die Attribute einer Klasse stehen außerhalb der Methoden der Klasse.
- ◆ Klassen können weitere „innere“ Klassen enthalten.
- ◆ Es werden Klassen programmiert, von denen dann die Objekte gebildet werden.
- ◆ Objekte sind „Instanzen“ einer Klasse.

## Grundsätzlicher Aufbau (3/3)

- ◆ ein Block besteht aus:
  - öffnender und schließender geschweifeter Klammer
  - Vereinbarungen (Definition von Variablen und Feldern)
  - Anweisungen bzw. Kontrollstrukturen
  - weiteren Blöcken (Blöcke können geschachtelt werden)

```
Syntax eines      {  
Blockes:         [ { Vereinbarung } ]  
                 [ { Anweisung } ]  
                 [ { Block } ]  
                 }
```

## 5.3 Elementare Datentypen

- ◆ **ganzzahlige Datentypen**                      **zugehöriges Objekt (Wrapper-Klasse)**
  - byte      1 Byte                      -128 ... +127                      Byte
  - short     2 Byte                      -2<sup>15</sup> .. +2<sup>15</sup> -1                      Short
  - int        4 Byte                      -2<sup>31</sup> .. +2<sup>31</sup> -1                      Integer
  - long      8 Byte                      -2<sup>63</sup> .. +2<sup>63</sup> -1                      Long
  - char      2 Byte (Unicode)                      Character
  
- ◆ **Gleitkommazahlen**
  - float     4 Byte                      7 Stellen Mantisse                      Float
  - double   8 Byte                      15 Stellen Mantisse                      Double
  
- ◆ **Wahrheitswert**
  - boolean   1 Bit (true oder false)                      Boolean

## 5.4 Vereinbarungen

- ◆ **Elementare Daten sind keine Objekte!**
- ◆ **Vereinbarungen elementarer Datentypen**
  - reservieren Speicherplatz entsprechend der Größe der Datentypen
  - verschiedene Variablen gleichen Typs können durch Komma getrennt aufgezählt werden
  - jede Vereinbarung wird durch Semikolon abgeschlossen
  - Variablen können (und sollten) eine Anfangsbelegung erhalten, um eine Fehlermeldung „**variable . . . might not have been initialized**“ beim Compilieren zu vermeiden!

## 5.5 Arithmetische Operatoren

- ◆ Für numerische Datentypen (auch Zeichen)
- ◆ „+“ - Addition, „-“ - Subtraktion, „\*“ - Multiplikation
- ◆ „/“ - Division (bei ganzen Zahlen wird abgerundet)
- ◆ zusätzlich bei ganzzahligen Datentypen
- ◆ „%“ - Restdivision (modulo)
- ◆ „++“ - Inkrement (Bsp.: i++, ++i)
- ◆ „--“ - Dekrement (Bsp.: j--, --j)

**Bem.:** i++ und ++i ist nicht dasselbe!

## 5.6 Zuweisungen, Bedingungsoperator

- ◆ „=“ - Zuweisung: der links vom Gleichheitszeichen stehenden Variable wird der rechts vom Gleichheitszeichen stehende Ausdruck zugewiesen
- ◆ „+=“, „-=“, „\*=“, „/=“ und „%=“ verkürzte Schreibweise, wenn sich der Wert der Variablen aus dem alten Wert dieser Variablen +, -, \*, /, und % einem Ausdruck ergibt Diese Schreibweise gilt auch für die logischen Verschiebe-Operatoren.
- ◆ Bedingungsoperator (bedingter Ausdruck) ist der einzige dreistellige Operator

### Syntax:

**Vergleichsausdruck ? Ausdruck\_1 : Ausdruck\_2**

## 5.7 Vergleichs- und logische Operatoren

### ◆ Vergleichsoperatoren

- < kleiner, <= kleiner gleich
- > größer, >= größer gleich
- == gleich, != ungleich

### ◆ logische Operatoren

- && logisches „und“ (and); Kurzauswertung
- & alle Operanden werden ausgewertet
- || logisches „oder“ (or); Kurzauswertung
- | alle Operanden werden ausgewertet
- ! logisches „nicht“ (not)
- ^ logisches „exklusives oder“ (xor)

## 5.8 Ausdruck, Anweisung

### ◆ Ausdruck

- beliebiger arithmetischer oder logischer Ausdruck
- Zuweisung oder Methodenaufruf (jeweils ohne Semikolon als Abschluss)

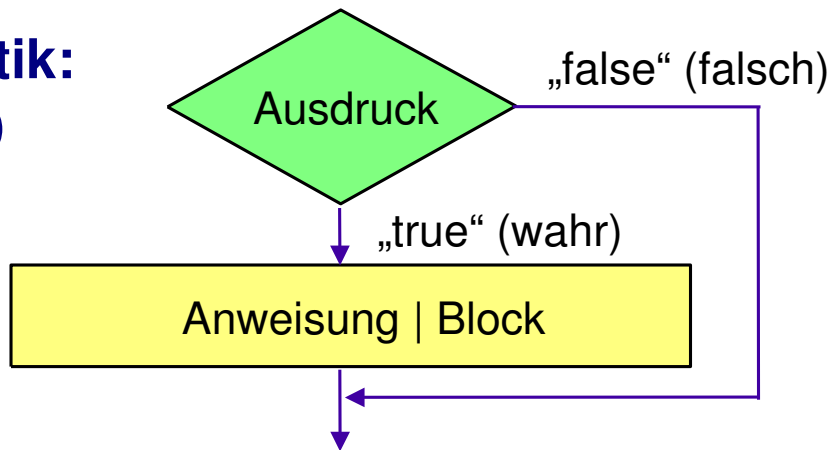
### ◆ Anweisung

- Zuweisung oder Methodenaufruf mit Semikolon als Abschluss
- Kontrollstruktur
- Ein Semikolon allein ist die leere Anweisung!

### Syntax der if - Verzweigung:

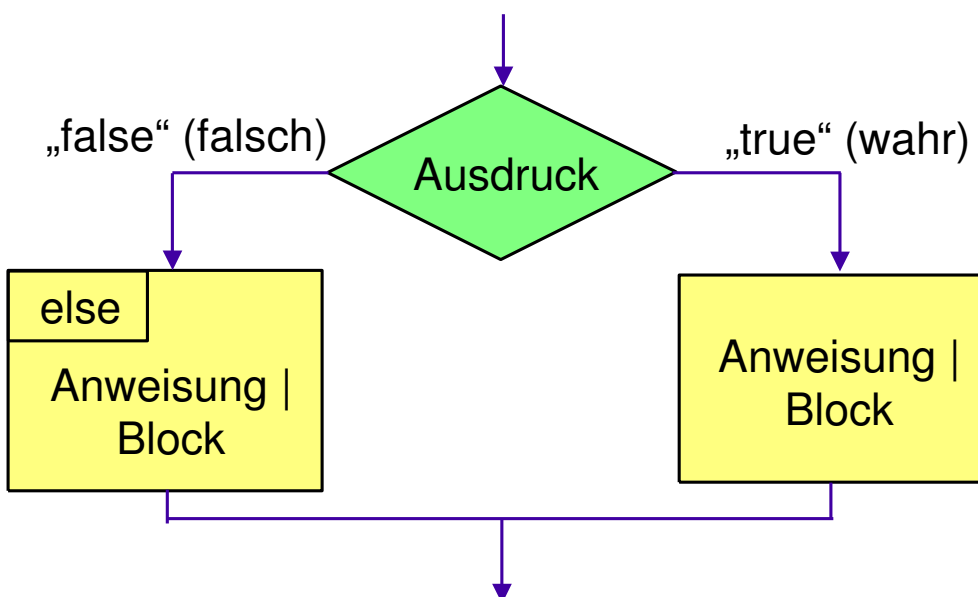
```
if (logischer Ausdruck)  
    Anweisung | Block  
[ else  
    Anweisung | Block ]
```

### Semantik: (1. Fall)



### **if**- Verzweigung

### Semantik der if-Verzweigung (2. Fall mit „else“):



## 6.2 for-Schleife (1/4)

### Syntax der for - Schleife:

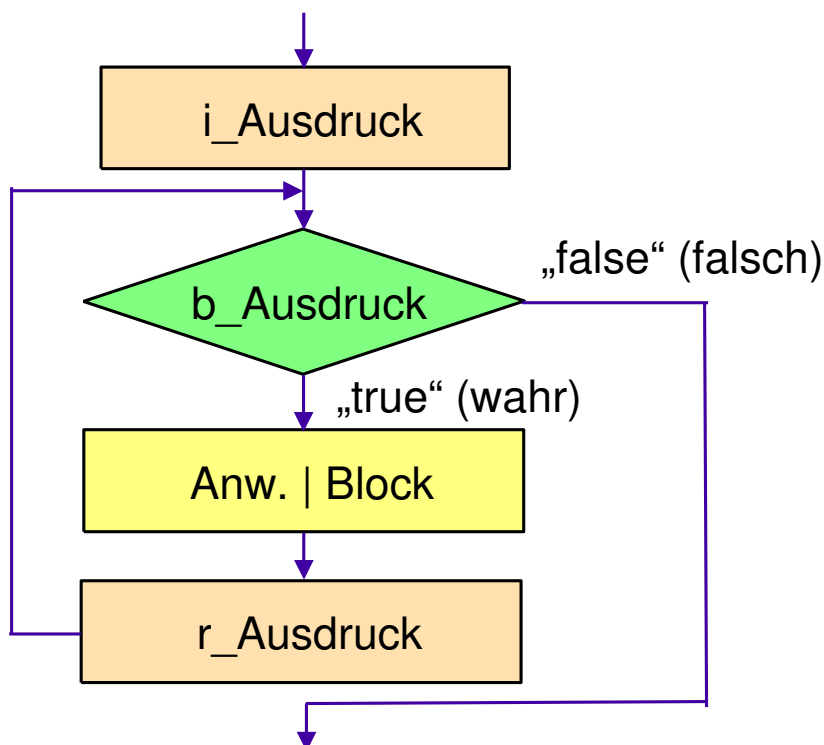
```
for ( [ i_Ausdruck ] ; b_Ausdruck ; [ r_Ausdruck ] )  
    Anweisung | Block
```

i\_Ausdruck: Initialisierung (beliebiger Ausdruck)  
b\_Ausdruck: Bedingung (logischer Ausdruck)  
r\_Ausdruck: Reinitialisierung (beliebiger Ausdruck)

**Beispiel:**     for (; i < 10; );     ist zulässig!

## for-Schleife (2/4)

### Semantik der for-Schleife:



## for-Schleife (3/4)

### Syntax der foreach - Schleife:

Vereinfachte Variante für listenartige Datenstrukturen wie Arrays und Collections (seit Java 5.0)

```
for (formaler Parameter : Ausdruck)  
    Anweisung | Block
```

formaler Parameter: Datentyp und Variablenname

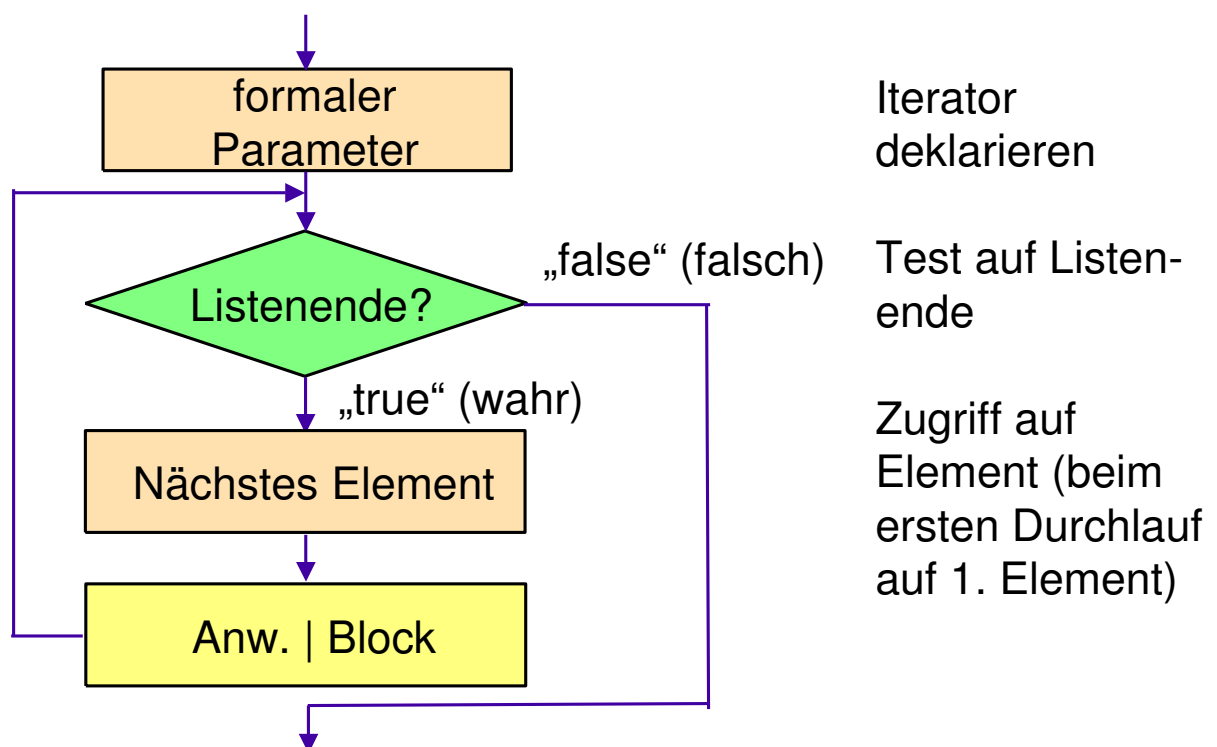
„:“ wird wie „in“ gelesen

Ausdruck: wirkt wie Iterator

```
Beispiel:    int [] args;  
             for (int a : args)  
             ...
```

## for-Schleife (4/4)

### Semantik der foreach-Schleife:

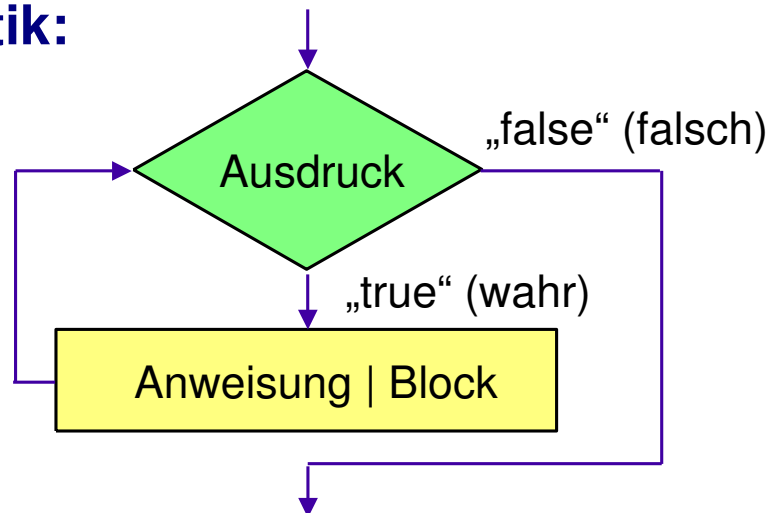


## 6.3 while-Schleife

### Syntax der kopfgesteuerten while-Schleife:

```
while (logischer Ausdruck)  
    Anweisung | Block
```

### Semantik:

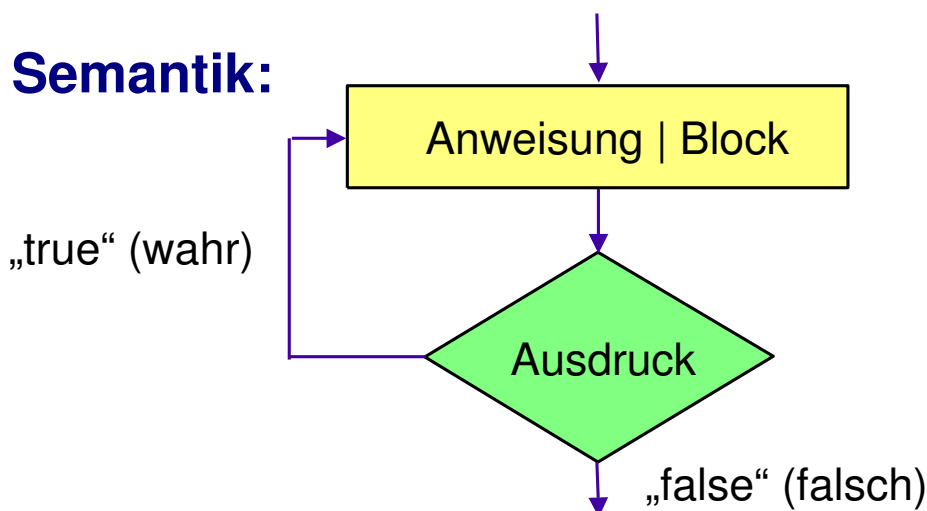


## while-Schleife

### Syntax der fußgesteuerten while-Schleife:

```
do  
    Anweisung | Block  
while (logischer Ausdruck);
```

### Semantik:



## 6.4 **switch**-Mehrfachverzweigung

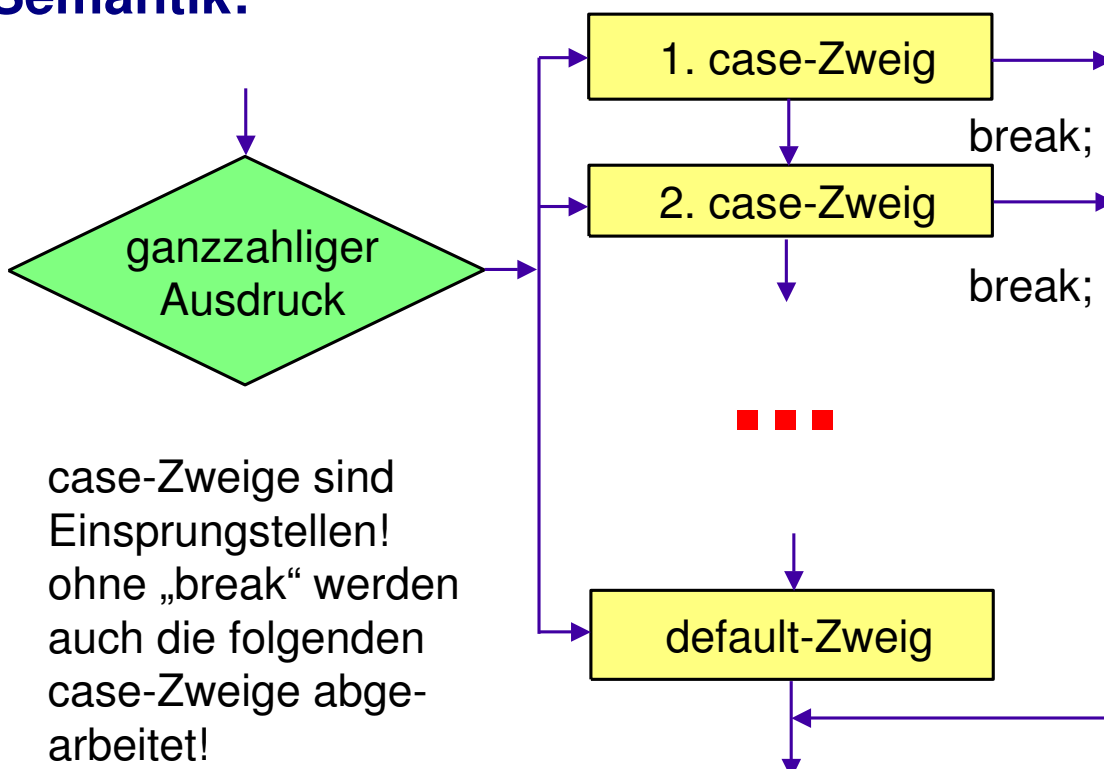
### Syntax der **switch**-Struktur:

```
switch (ganzzahliger Ausdruck)
{
    case ganzz. Konstante: [Anweisung [en]]
    {
        [Block]
        [break;]
    }

    default: [Anweisung [en]]
             [Block]
}
}
```

## **switch**-Mehrfachverzweigung

### Semantik:



## 6.5 Schlüsselwörter **continue** und **break**

- ◆ **continue**
  - nur für Schleifen „for“, „while“ und „do ... while“ gültig
  - der aktuelle Schleifendurchlauf wird abgebrochen, die Reinitialisierung wird ausgeführt (bei der for-Schleife) und es wird zum Schleifentest gesprungen
- ◆ **break**
  - nur für Schleifen „for“, „while“, „do ... while“ und „switch“
  - nur die unmittelbar umgebende Schleife oder „switch“ wird verlassen
- ◆ **continue und break mit Sprungmarken sind erlaubt**

## 6.6 goto, return und exit

- ◆ **goto**
  - **Schlüsselwort, aber nicht erlaubt!**
- ◆ **return**
  - Rückkehr in unmittelbar aufrufende Methode (keine Methode!)
- ◆ **Syntax:** `return [ Ausdruck ];`
- ◆ **System.exit (ganzzahliger Ausdruck);**
  - Standardmethode!
  - Liefert Rückkehrwert an das Betriebssystem

# Literaturangaben

- [1] Java 2 SDK v 1.2.2, *Grundlagen Programmierung*  
HERDT-Verlag für Bildungsmedien GmbH, Nackenheim
- [2] Guido Krüger: *Handbuch der Java-Programmierung*,  
Addison-Wesley Verlag,  
ISBN 3-8273-2201-4
- [3] Brit Schröter: *KompaktReferenz Java2*,  
DATA BECKER GmbH & Co. KG,  
ISBN 3-8158-1477-4
- [4] *Internet HTML-Seiten*,  
<http://java.sun.com>

