BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

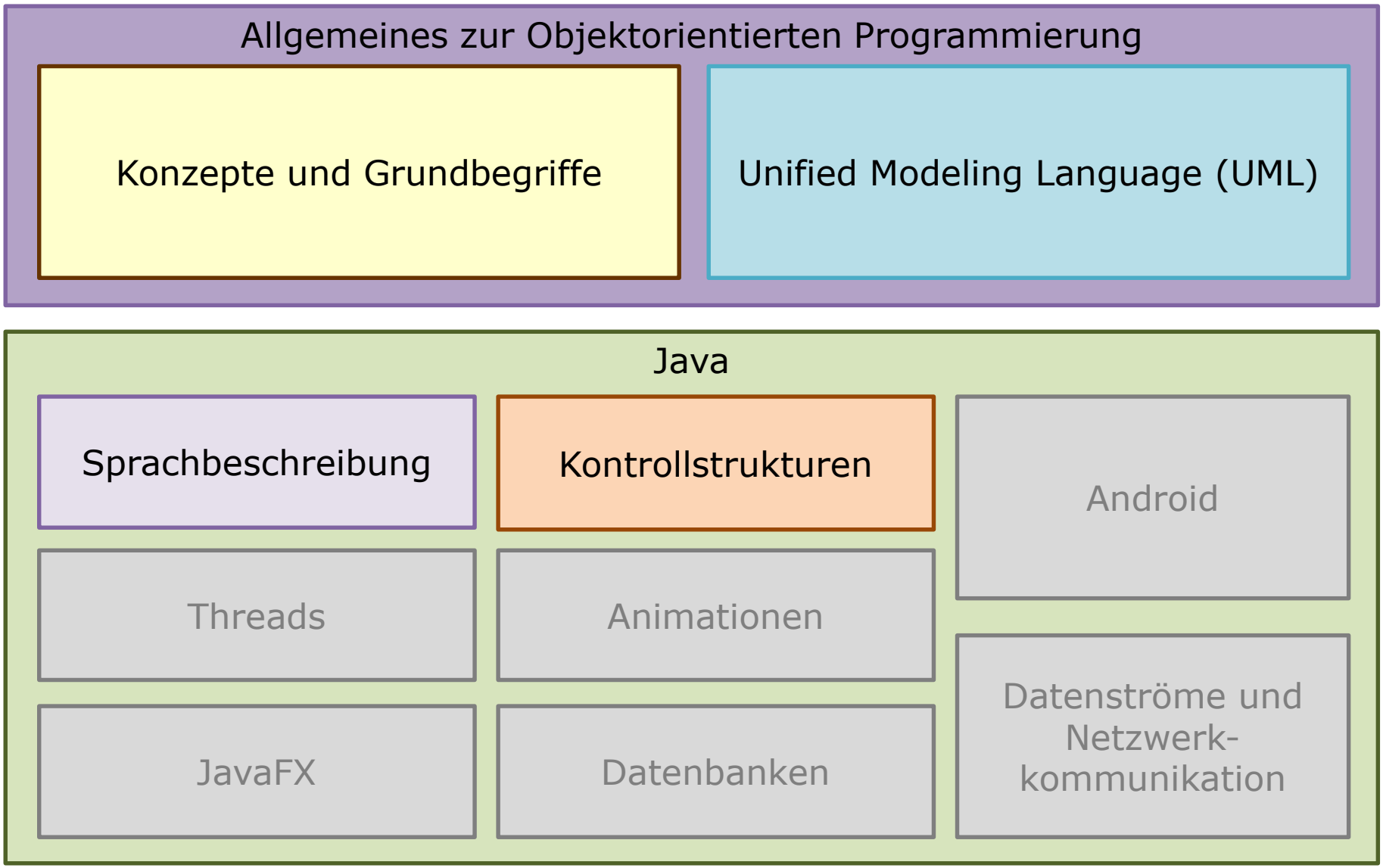
Objektorientierte Programmierung

Java

mit Material von Prof. E. Engelhardt

Prof. Dr.-Ing. Tenshi Hara
tenshi.hara@ba-sachsen.de

Aufbau der Lehrveranstaltung



Abkürzungen

API	Application Programming Interface
AWT	Abstract Window Toolkit (oft auch Abstract Windowing 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

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“ (Object Application Kernel)
- „Oak“ (Eiche) gab es schon, deshalb „Java“
- „JavaSoft“ ist Tochterfirma von Sun Microsystems
- 2010 wurde Sun Microsystems von Oracle übernommen

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 von Webrunner 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

Historie der Java-Entwicklungsstufen (1/7)

- JDK 1.0 (Java Development Kit 1.0) Jahr 1996
  - Grundlegende Objekte (Objekte, Strings, Threads, Ein- und Ausgabe, Datenstrukturen, Systemeigenschaften)
  - Grafikprogrammierung (**A**bstract **W**indow **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, Datenbankabbindung
  - RMI (Remote Method Invocation), Reflections
  - JNI (Java Native Interface)
  - mathematische Hilfsklassen (java.math)

Historie der Java-Entwicklungsstufen (2/7)

- 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.)

Historie der Java-Entwicklungsstufen (3/7)

- 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

Historie der Java-Entwicklungsstufen (4/7)

- Version 5.0 der Java 2 Standard Edition („Tiger“)
  - 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

Java-Entwicklungsstufen (5/7)

- JSE 7 – Dolphin
  - neue Filesystem-API (NIO.2)
  - Netzwerkprotokolle (IPv6)
  - Unicode-Update (Unicode 6.0)
  - Look-and-Feel Nimbus
  - String in switch-Anweisung
  - JavaFX
  - …
- JSE 8 (älteste LTS-Version)
  - neue Date- und Time-API
  - Performance-Verbesserungen (Parallelisierung)
  - Lambda-Ausdrücke
  - …

Java-Versionen mit und ohne Support

JDK 1.0		1996
JDK 1.1		1997
JDK 1.1.4	Sparkler	12.09.1997
JDK 1.1.5	Pumpkin	03.12.1997
JDK 1.1.6	Abigail	24.04.1998
JDK 1.1.7	Brutus	28.09.1998
J2SE 1.2	Playground	04.12.1998
J2SE 1.2.1		30.03.1999
JDK 1.1.8	Chelsea	08.04.1999
J2SE 1.2.2	Cricket	08.07.1999
J2SE 1.3	Kestrel	08.05.2000
J2SE 1.3.1	Ladybird	17.05.2001
J2SE 1.4.0	Merlin	13.02.2002
J2SE 1.4.1	Hopper	16.09.2002
J2SE 1.4.2	Mantis	26.06.2003
J2SE 5.0	Tiger	29.09.2004
JSE 6	Mustang	11.12.2006
JSE 7	Dolphin	28.07.2011
JSE 8	8 LTS	18.03.2014
JSE 9		27.07.2017
JSE 10	18.3	20.03.2018
JSE 11 LTS	18.9	25.09.2018
JSE 12	19.3	19.03.2019
JSE 13	19.9	17.09.2019
JSE 14	20.3	17.03.2020
JSE 15	20.9	15.09.2020
JSE 16	21.3	16.03.2021
JSE 17 LTS	21.9	Sept. 2021

Java-Entwicklungsstufen (6/7)

- JSE 9
  - neues Support-Modell (Support nur bis zum Erscheinen des Nachfolgers)
  - Jigsaw-Modulsystem
  - Integration der jshell
- JSE 10
  - halbjährliches Non-LTS-Update
- JSE 11
  - halbjährliches Update mit LTS
- JSE 12 bis JSE 16
  - halbjährliches Non-LTS-Update
- JSE 17
  - halbjährliches Update mit LTS

Java-Versionen mit und ohne Support

JDK 1.0		1996
JDK 1.1		1997
JDK 1.1.4	Sparkler	12.09.1997
JDK 1.1.5	Pumpkin	03.12.1997
JDK 1.1.6	Abigail	24.04.1998
JDK 1.1.7	Brutus	28.09.1998
J2SE 1.2	Playground	04.12.1998
J2SE 1.2.1		30.03.1999
JDK 1.1.8	Chelsea	08.04.1999
J2SE 1.2.2	Cricket	08.07.1999
J2SE 1.3	Kestrel	08.05.2000
J2SE 1.3.1	Ladybird	17.05.2001
J2SE 1.4.0	Merlin	13.02.2002
J2SE 1.4.1	Hopper	16.09.2002
J2SE 1.4.2	Mantis	26.06.2003
J2SE 5.0	Tiger	29.09.2004
JSE 6	Mustang	11.12.2006
JSE 7	Dolphin	28.07.2011
JSE 8	8 LTS	18.03.2014
JSE 9		27.07.2017
JSE 10	18.3	20.03.2018
JSE 11 LTS	18.9	25.09.2018
JSE 12	19.3	19.03.2019
JSE 13	19.9	17.09.2019
JSE 14	20.3	17.03.2020
JSE 15	20.9	15.09.2020
JSE 16	21.3	16.03.2021
JSE 17 LTS	21.9	Sept. 2021

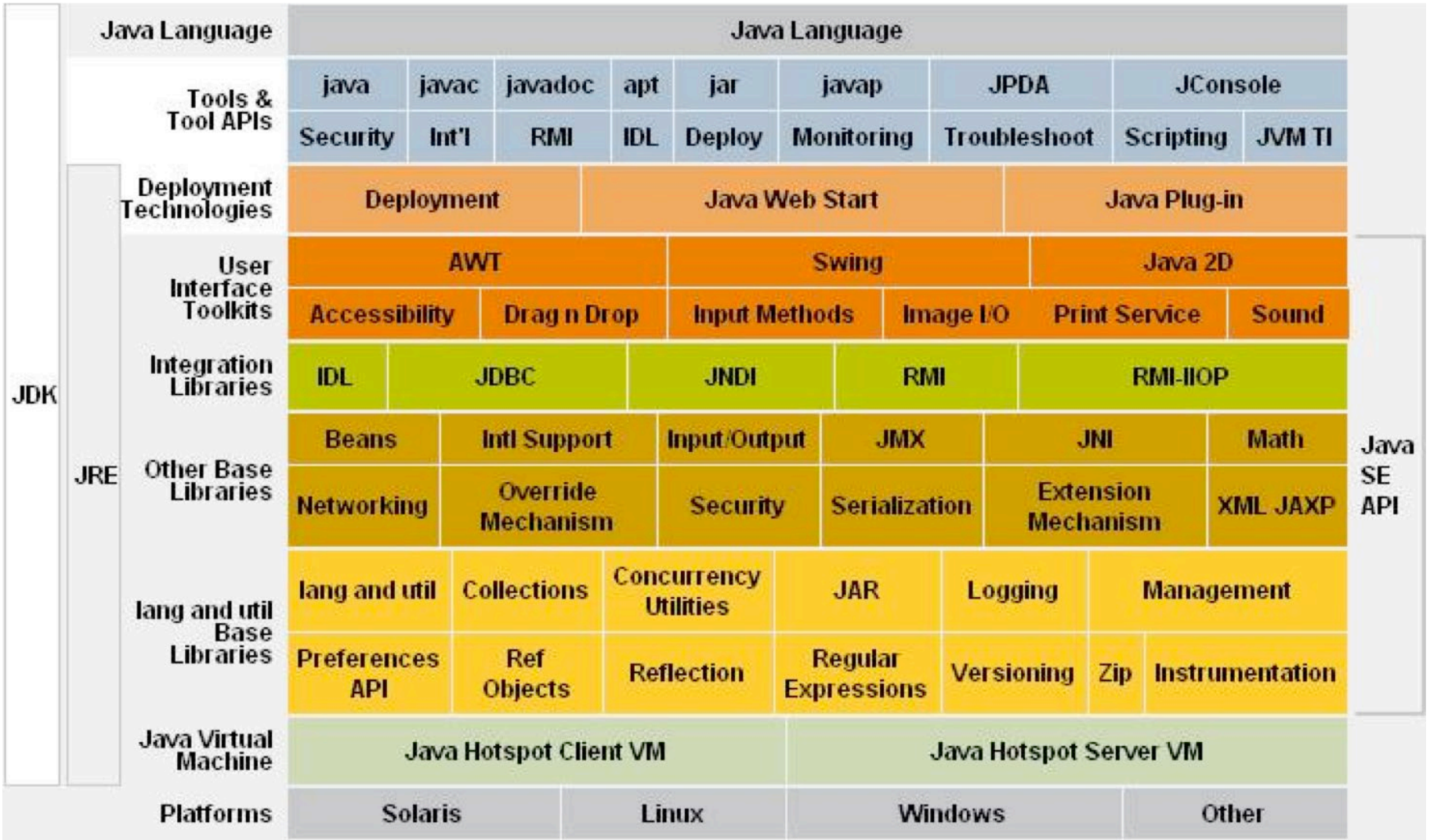
Historie der Java-Entwicklungsstufen (7/7)

kategorisierte Übersicht aller Änderungen
an Java und der JVM seit JDK 8:

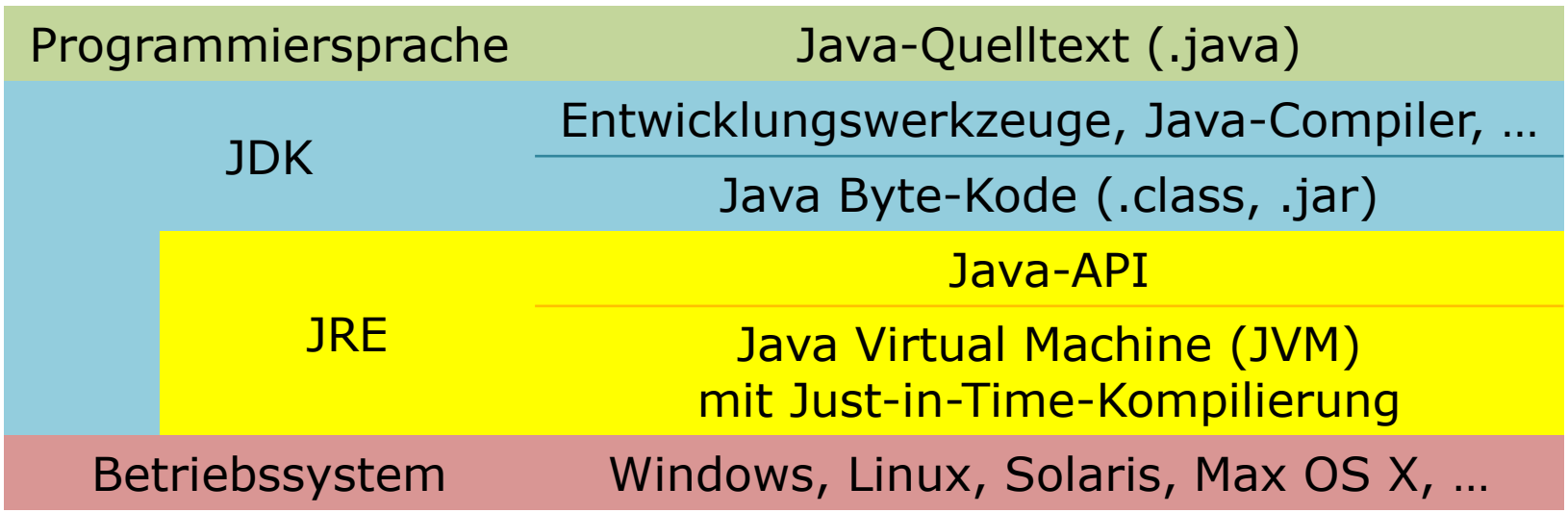
https://advancedweb.hu/a-categorized-list-of-all-java-and-jvm-features-since-jdk-8-to-16/

Architektur von Java

ARCHITEKTUR (1/2)



Architektur (2/2)



Standardpakete seit Java 6 (1/2)

• java.applet	Applets
• java.awt	Abstract Window 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 seit Java 6 (2/2)

- 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

Java und JavaScript

- JavaScript ist Makro-Programmiersprache
- Entwicklung der Firma Netscape
- JavaScript wird nicht übersetzt und ist im Quelltext in der HTML- Seite eingefügt (`<SCRIPT> ... </SCRIPT>`)
- objektbasierend (nicht streng objektorientiert)
- im Vergleich der Möglichkeiten eine unechte Untermenge von Java

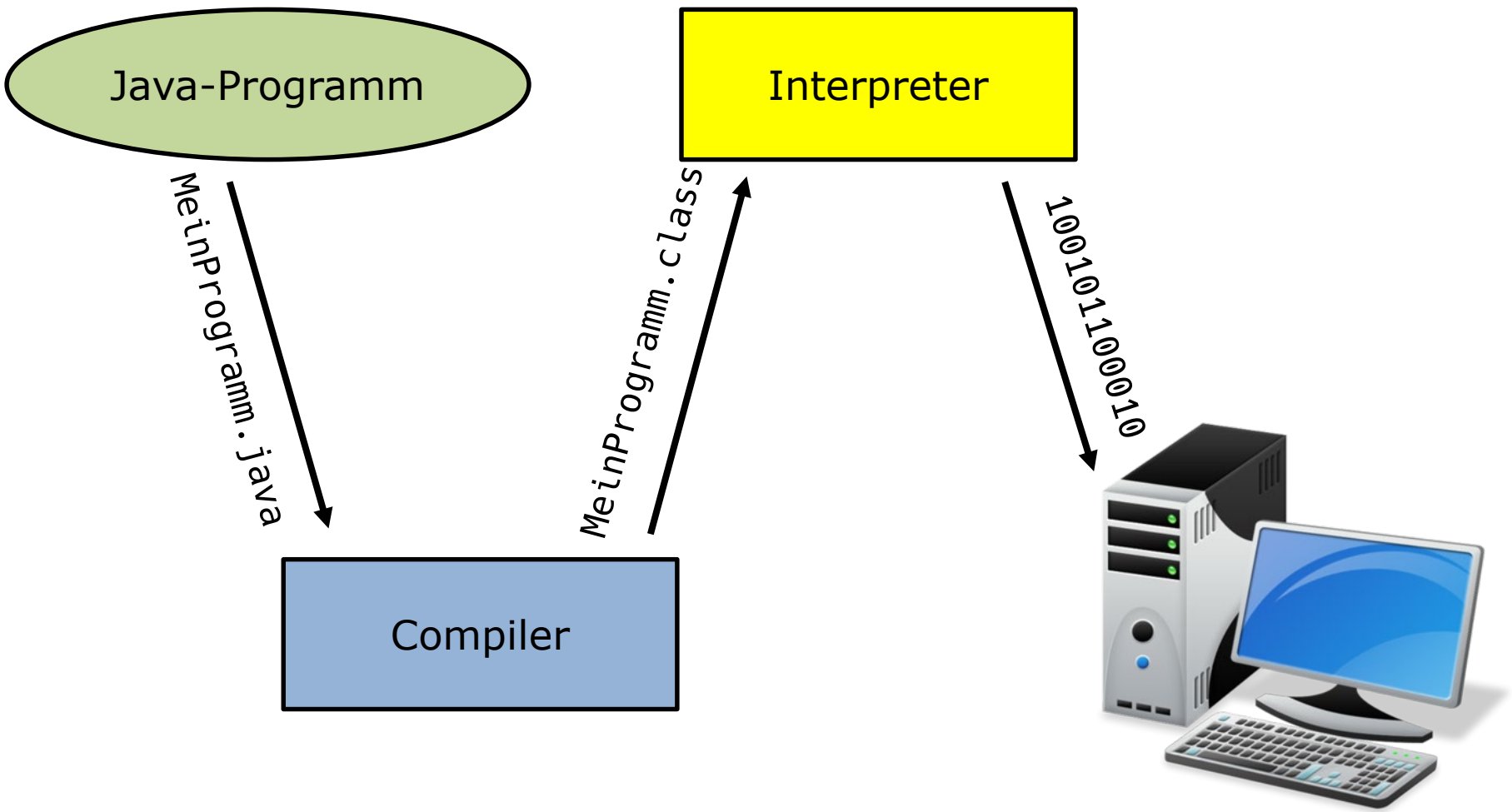
Was ist Java?

Java ist zweierlei: eine Programmiersprache und eine Plattform

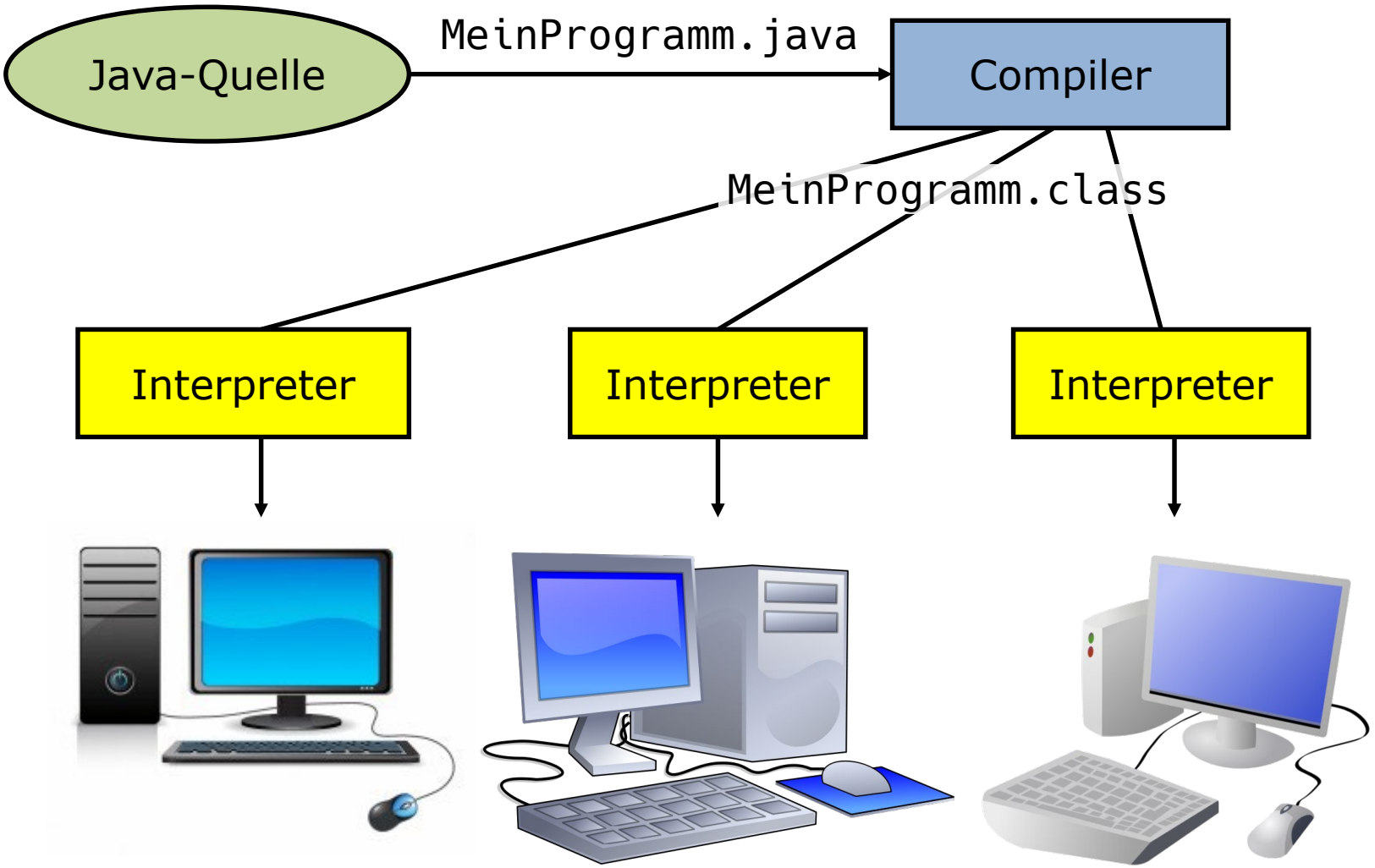
Eigenschaften von Java als Sprache:

- einfach
- architekturneutral
- objektorientiert
- portabel
- verteilt
- Interpretersprache
- multithreaded
- hohe Performance
- robust
- dynamisch
- sicher

Java als Programmiersprache



WRITE ONCE, RUN EVERYWHERE



Was kann Java?

Java Programme werden unterschieden in:

- Application ⇒ eigenständige Programme
- Applet ⇒ in anderen Programm (z.B. Webbrowser) laufende Programme
- Server ⇒ Webserver, Mailserver, ...
- Servlet ⇒ ähnlich einem Applet, eine Art Server-Erweiterung
- JSP Java Server Page ⇒ entspricht ASP

JAVA-API (1/2)

Jede vollständige Java-Implementation enthält 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
• Security	Elektronische Unterschriften, private/öffentliche Schlüssel, Zugriffskontrolle, Zertifikate
• Software Components	JavaBeans, die sich auch in existierende Komponentenarchitekturen einklinken können (z.B. OpenDoc, OLE/COM/Active-X)
• Object Serialization	Kommunikation via RMI
• JDBC	Einheitlicher Zugriff auf ein breites Spektrum relationaler Datenbanken

JAVA-API (2/2)

zusätzlich zur Core-API gibt es noch Erweiterungen

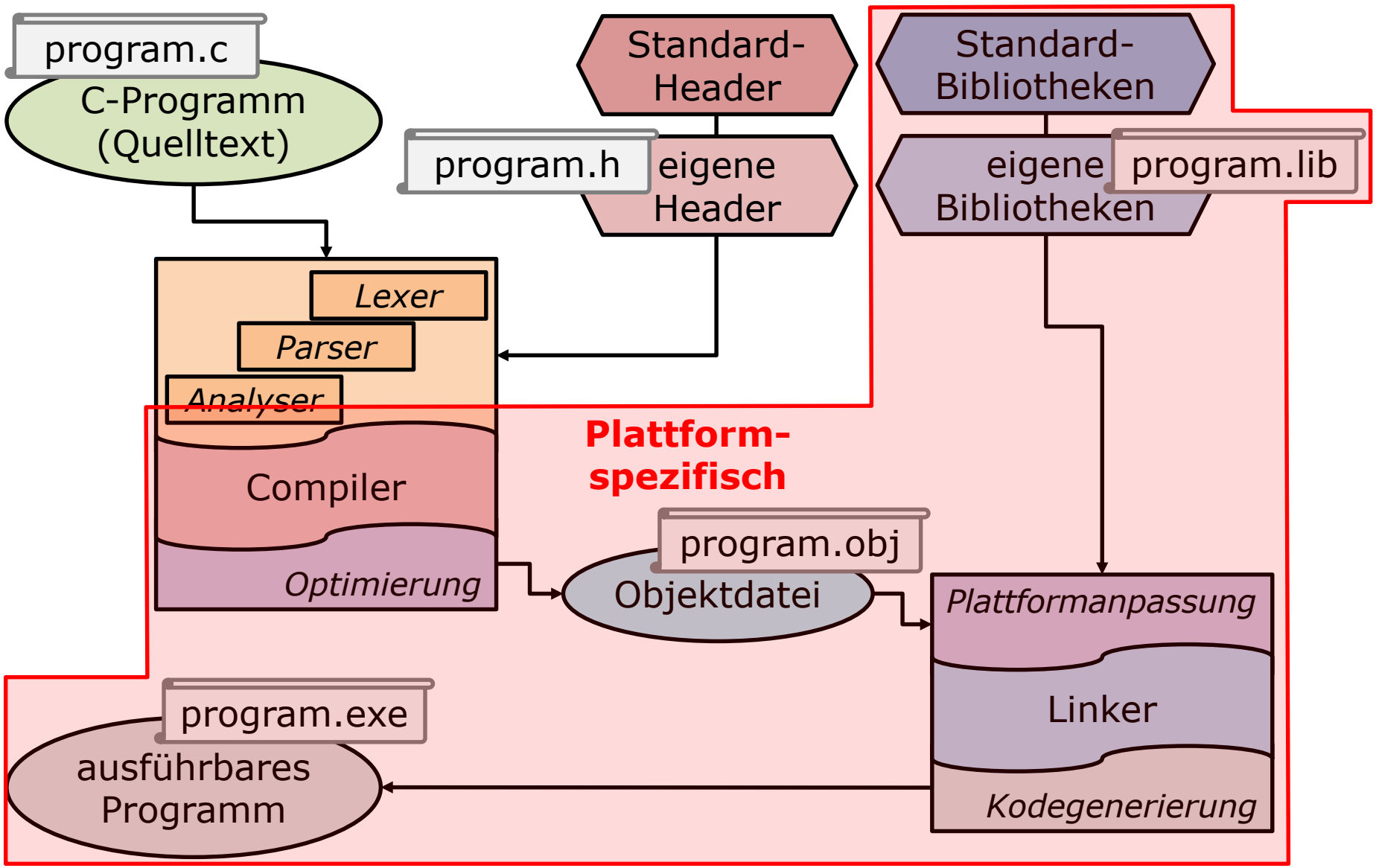
- Java-3D
- JAI (Java Advanced Imaging)
- JMF (Java Media Framework)
- Telefonie
- Server
- Sprachausgabe
- Animationen
- ...

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
- Write once, run everywhere
- Anwendungen leichter verteilen
- ...

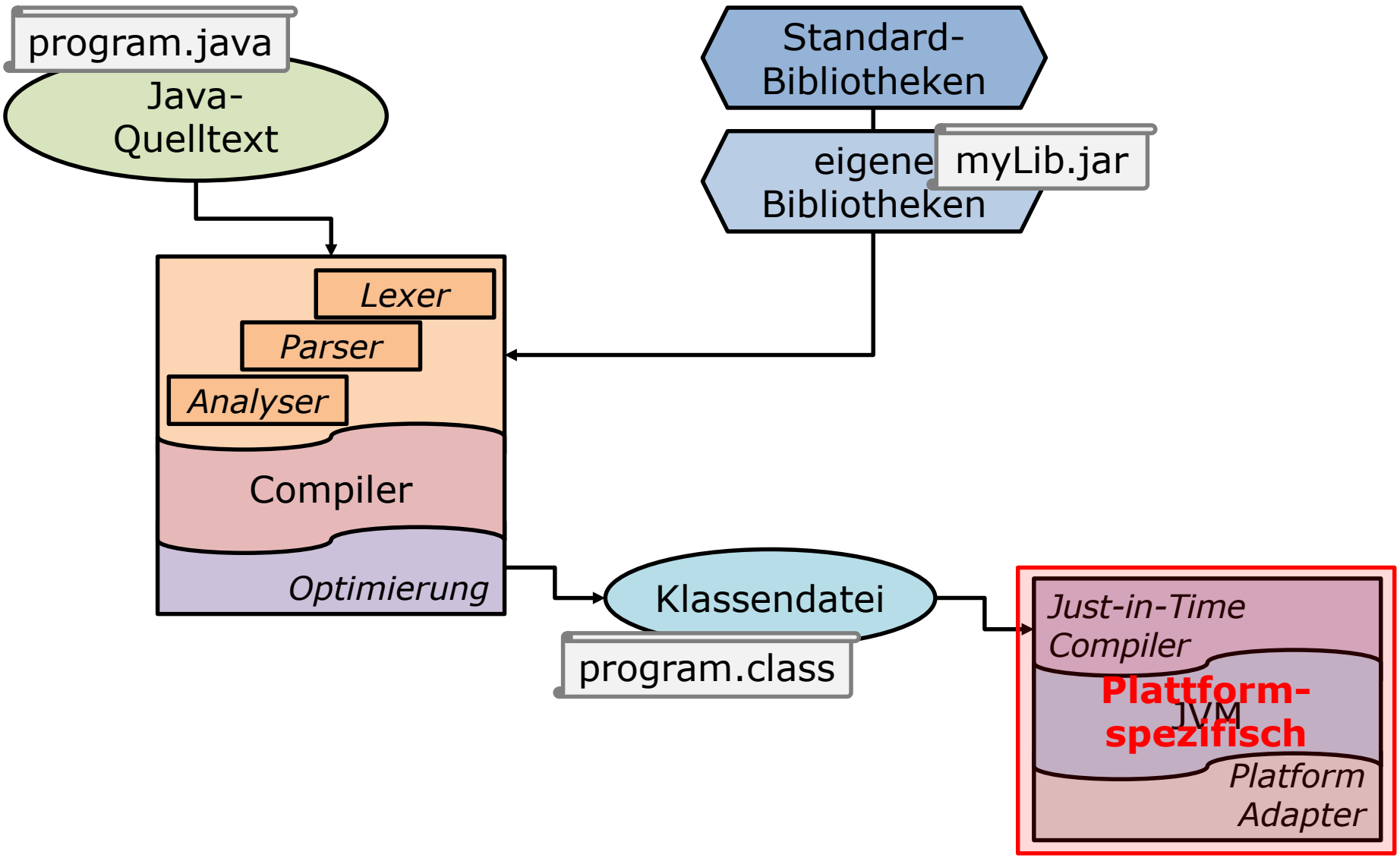
Wiederholung: Ein C-Programm ausführen (1/2)



Wiederholung: Ein C-Programm ausführen (2/2)

- C-Quellcode muss auf jeder Plattform mit dem dort verfügbaren Compiler neu übersetzt werden
- Bibliotheken müssen für die jeweilige Zielplattform (Prozessor und Betriebssystem) erstellt sein
- Linker erstellt ausführbares Programm für die entsprechende Plattform
- Programm nur auf Binärkode-kompatiblen Plattformen ausführbar
- mit ANSI-C kann zumindest Quellcode-Kompatibilität erreicht werden

Ein Java-Programm ausführen (1/2)



Ein Java-Programm ausführen (2/2)

- Java-Quellkode wird durch Java-Compiler in einen maschinenkodenahen, plattformunabhängigen Bytekode übersetzt
- Java-Bytekode wird mittels JVM (Java Virtual Machine) auf der jeweiligen Zielplattform ausgeführt
- Bytekode muss nur einmal erstellt werden und ist auf jeder Zielplattform mittels der dort vorhandenen JVM ausführbar

Applikationen vs. Applets

- 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

- 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

Allgemeines zu Anwendungen und Klassen

ERSTES PROGRAMM

```
import java.io.*;

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

Grundgerüst einer Java-Applikation

- Applikation besteht aus mindestens einer `public`-Klasse, welche die Methode `main` enthält
- Name der `public`-Klasse ist identisch mit dem Dateinamen (ohne Dateiendung)
- Applikation kann mehrere `public`-Klassen mit (unterschiedlichen) `main`-Methoden enthalten
- 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(…)` ist `import java.io.*;` nicht erforderlich

Java-Programmentwicklung ohne IDE
(nur zur Veranschaulichung)

Editieren des Java-Quelltextes mit beliebigem Editor

```
tenshi ~ $ pico HalloWelt.java
tenshi ~ $ javac HalloWelt.java
tenshi ~ $ java HalloWelt
Hallo Welt!
tenshi ~ $
```

Java-Bytekode wird erzeugt und in der Datei `HalloWelt.class` gespeichert

Java-Bytekode wird der JVM übergeben und ausgeführt

Erstes Applet (1/2)

```java
/* 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);
   }
}
```

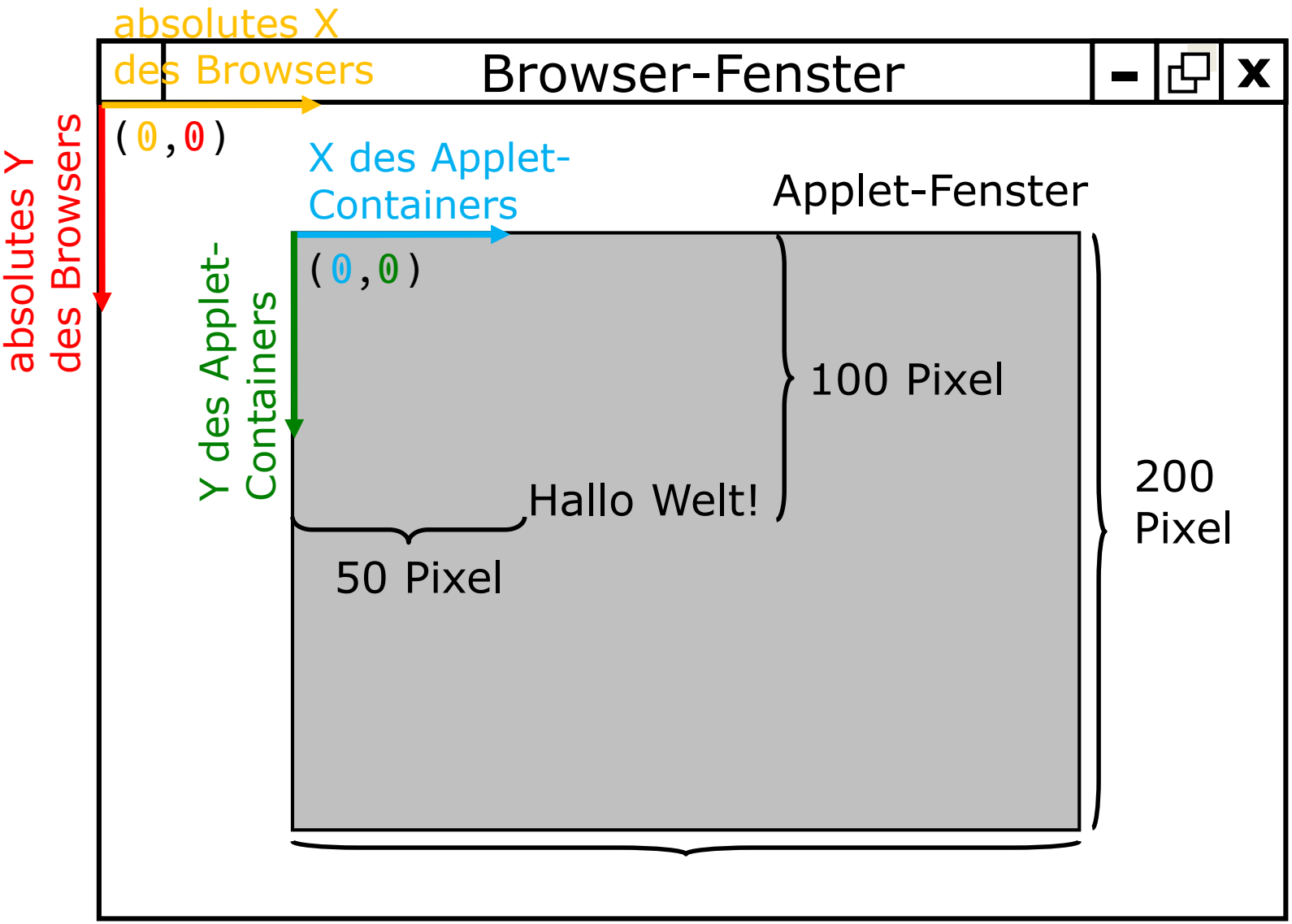
ERSTES APPLET (2/2)

```
<!-- HalloWeltApplet.html -->
<!-- … -->
<object classid="java:HalloWeltApplet.class"
   codetype="application/java-vm"
   style="width:300px; height:200px">
      Java erforderlich!<br />
      Die Java-Runtime-Umgebung k&ouml;nnen Sie
      <a href="https://java.com/de/download/"
         target="_blank">hier downloaden</a>.
</object>
<!-- … -->
```

Applet-Grundgerüst

- Applet besteht aus mindestens einer `public`-Klasse ohne `main`-Methode
- Name der `public`-Klasse ist identisch mit dem Dateinamen (ohne Dateiendung)
- als Teil eines Fensters (Panel) einer übergeordneten grafikorientierten Anwendung (AppletViewer oder Browser) ist ein Applet automatisch grafikorientiert
- Applet läuft in Sandbox
- zur Ausführung eines Applets muss immer eine entsprechende HTML-Datei vorhanden sein
- in der `paint`-Methode erfolgen die Grafikausgaben

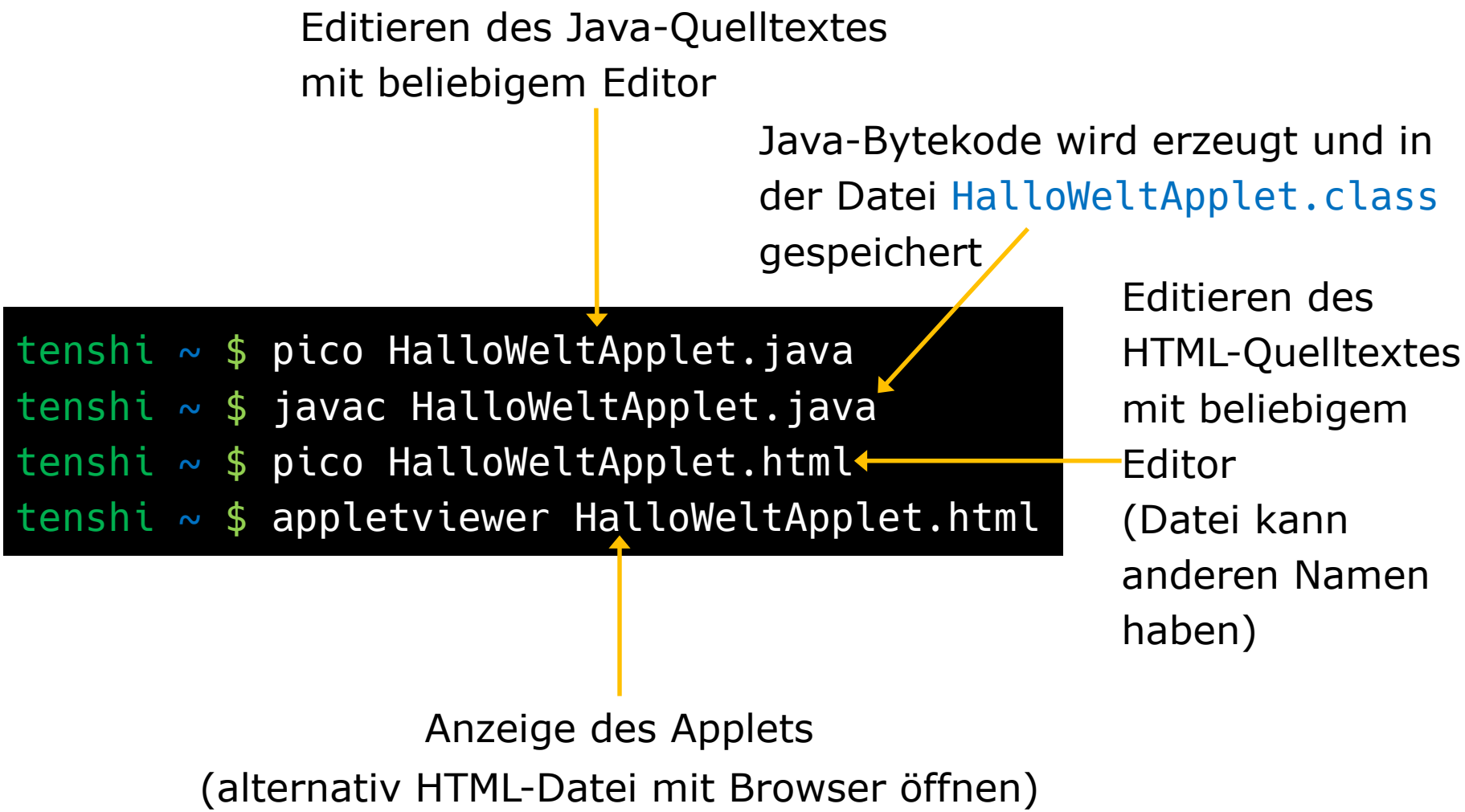
Koordinatensystem und Einheiten



Hinweise zu Applets

- Lage des Applets im Browserfenster wird durch die HTML-Tags bestimmt (Table, Frame, ...)
- Größe des Applets steht im Applet-Tag (`width` und `height`)
  (Angaben, wenn nicht anders angegeben, in Pixel)
- die meisten Browser stellen Applet-Hintergrund standardmäßig grau dar
- verschiedene Browser können unterschiedlich reagieren!
- bei Ausführung mittels AppletViewer wird nur Applet-Fenster angezeigt
- zum erstellenden JDK gehöriger AppletViewer sollte das Applet auf jeden Fall fehlerfrei abarbeiten!

Java-Applet ohne IDE



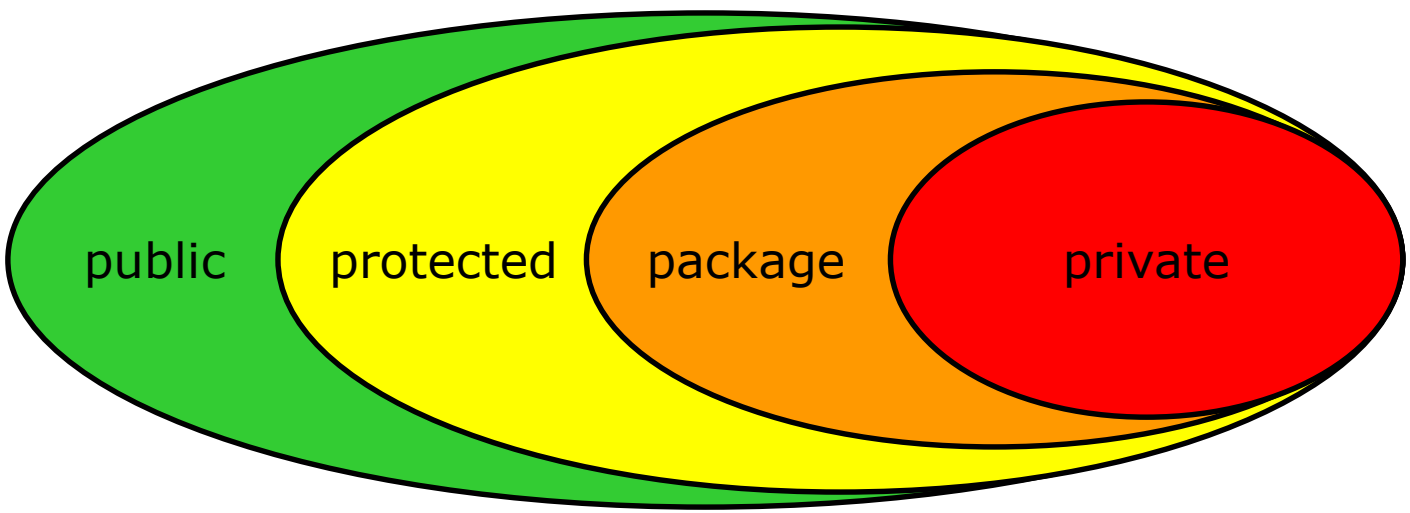
Allgemeine Form einer Klasse

```
public class KlassenName extends Superklasse {
   /*
    * Variablen sind öffentlich, privat, geschützt, ...
    * Variablennamen fangen mit Kleinbuchstaben an
    */
   protected typ variableName;
   private typ variableName;

   /*
    * Methoden sind öffentlich, privat, ...
    * Methodennamen fangen mit Kleinbuchstaben an
    */
   public typ methodenName(typ parameterName,...) {
      // Anweisungen
   }
}
```

Zugriffsberechtigungen

- private
  - nur die eigene Klasse kann auf Datum/Methode zugreifen
  - auch abgeleitete Klassen haben keinen Zugriff
- package: wie private, aber Klassen im Paket haben Zugriff
- protected: wie private, aber abgeleitete Klassen können zugreifen
- public: jeder kann auf Datum/Methode zugreifen



Pakete

Pakete (Java Packages)

- Zusammenfassung von Klassen und Schnittstellen (Interfaces) in logische Gruppen
- Bildung von eigenen Namensräumen
- definierter Export von Klassen bzw. Verbergen von Klassen (Information Hiding), die nicht exportiert werden sollen
- definierter Import von Klassen bzw. Fremdpaketen

Schlüsselwörter

- `package <Paketname>`: Deklaration für Java-Paket
- jede Java-Klasse, die zum Paket gehören soll, muss in einer Quelldatei stehen, die mit Schlüsselwort `package` gefolgt vom Paketnamen beginnt
- `import`: Importieren von Klassen, Interfaces und ganzen Paketen

Struktur von Java

Grundsätzlicher Aufbau (1/2)

Sammlung von Klassen, auch über mehrere Dateien (Module)

- mindestens eine Klasse, die genau so heißen muss wie die Datei (ohne Dateiendung `.java`) und `public` sein muss
- mindestens eine `main`-Methode in einer Applikation
- in `main` wird Instanz der Hauptklasse gebildet

Klassen bestehen aus

- Attributen (Variable, Objekte)
- Methoden (Funktionalität)
- evtl. Konstruktoren bei Applikationen
- den Methoden `init`, `start` und `stop` bei Applets

Grundsätzlicher Aufbau (2/2)

- außerhalb von Klassen gibt es keine Methoden
- Klassen sind die allgemeinen Beschreibungen
  - es werden Klassen programmiert, von denen dann die Objekte gebildet werden
  - Objekte sind Instanzen einer Klasse → Prozess des Bildens eines Objektes wird als Instanziierung bezeichnet
  - im Programm wird mit Objekten gearbeitet
- Attribute einer Klasse stehen außerhalb der Methoden der Klasse
- Klassen können weitere *innere* Klassen enthalten

Blöcke

ein Block besteht aus

- öffnender und schließender geschweifter Klammer
- Vereinbarungen (Definition von Variablen und Feldern)
- Anweisungen bzw. Kontrollstrukturen
- weiteren Blöcken (Blöcke können geschachtelt werden)

Syntax eines Blockes

```
<Block> = '{' ( (<Vereinbarung> | <Anweisung>) | <Block>) '}'
```

ELEMENTARE DATENTYPEN

ganzzahlige Datentypen			Wrapper-Klasse
• byte	1 Byte	-128 ... +127	Byte
• short	2 Byte	-2 15 .. +2 15 -1	Short
• int	4 Byte	-2 31 .. +2 31 -1	Integer
• long	8 Byte	-2 63 .. +2 63 -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

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 Warnung (`variable […] might not have been initialized`) beim Kompilieren zu vermeiden

ARITHMETISCHE OPERATOREN

- für numerische Datentypen (auch Zeichen)
  - `+` Addition
  - `-` Subtraktion
  - `*` Multiplikation
  - `/` Division (bei ganzen Zahlen wird abgerundet)
- zusätzlich bei ganzzahligen Datentypen
  - `%` Restdivision (modulo)
  - `++` Inkrement
  - `--` Dekrement

Achtung: `i++` und `++i` bzw. `i--` und `--i` sind nicht äquivalent!

Zuweisungen, Bedingungsoperator

= der links vom Gleichheitszeichen stehenden Variable wird der rechts vom Gleichheitszeichen stehende Ausdruck zugewiesen

- verkürzte Schreibweise, wenn sich der Wert der Variablen aus dem alten Wert dieser Variablen +, -, *, /, oder % einem Ausdruck ergibt
  - +=, -=, *=, /= bzw. %=
  - gilt auch für die logischen Verschiebeoperatoren

- Bedingungsoperator ist der einzige dreistellige Operator

```
<Bedingungsoperator> = <Vergleichsausdruck>
                       '?' <Ausdruck_1>
                       ':' <Ausdruck_2>
```

Vergleichsoperatoren, 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)

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 (`if`, `for`, `while`, `do…while` und `switch`)
- ein Semikolon allein ist die leere Anweisung!

IF-VERZWEIGUNG

```
<if-Anweisung> = "if" '(' <logischer Ausdruck> ')'
                    <Anweisung> | <Block>
                  [ "else"
                    <Anweisung> | <Block>
                  ]
```

FOR-SCHLEIFE

```
<for-Schleife> = "for" '('
                    [ <i_Ausdruck> ] ';'
                      <b_Ausdruck>   ';'
                    [ <r_Ausdruck> ]
                  ')'
                    <Anweisung> | <Block>
```

Dabei gilt

- `<i_Ausdruck>` Initialisierung (beliebiger Ausdruck)
- `<b_Ausdruck>` Bedingung (logischer Ausdruck)
- `<r_Ausdruck>` Reinitialisierung (beliebiger Ausdruck)

Achtung! Das ist zulässig: `for (;i < 10;);`

FOREACH-SCHLEIFE (1/2)

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

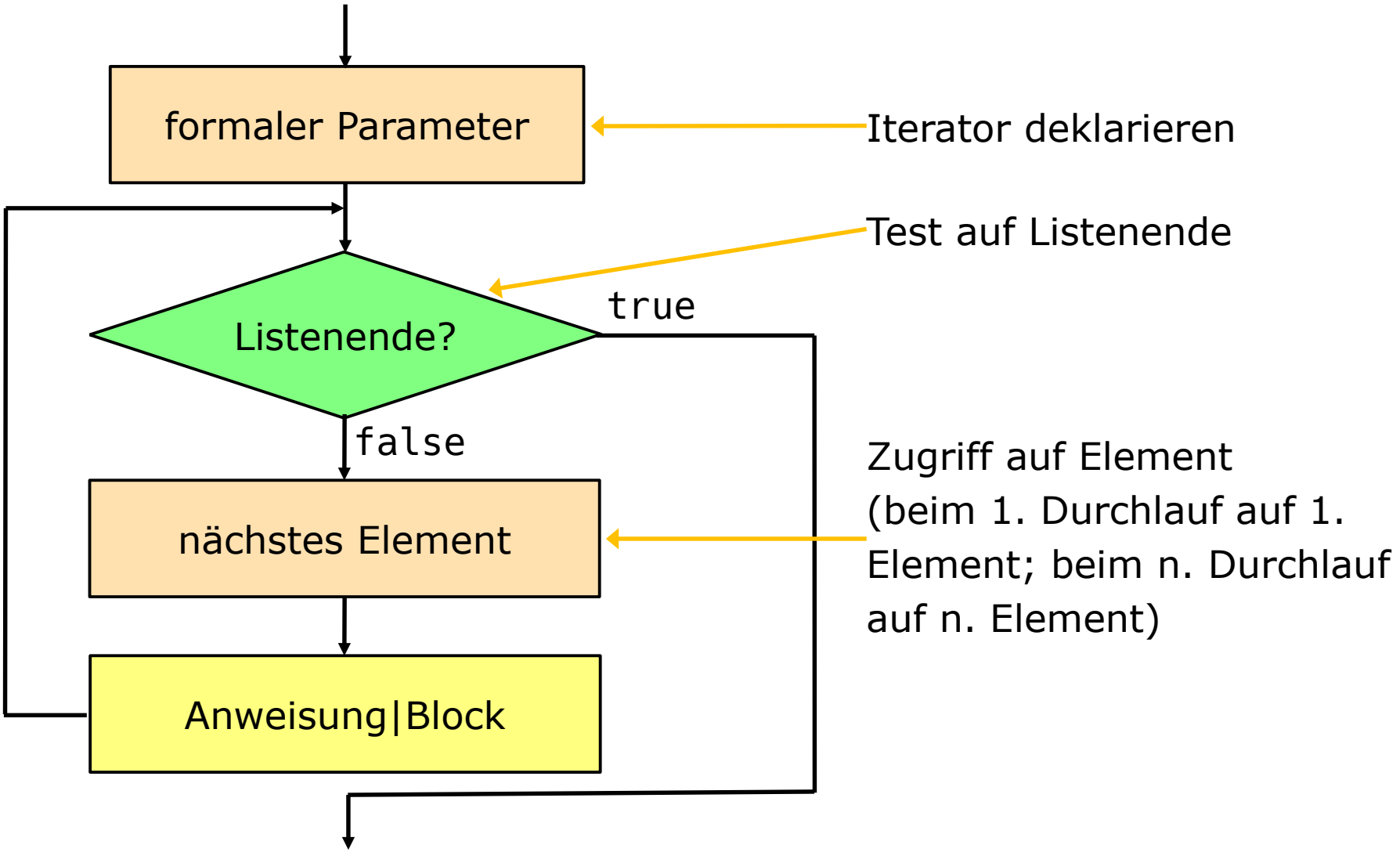
```
<foreach-Schleife> = "for" '('
                       <formaler Parameter> ':' <Ausdruck>
                     ')'
                       ( <Anweisung> | <Block> )
```

Dabei gilt

- `<formaler Parameter>` Datentyp und Variablenname
- `':'` wird wie „in" gelesen
- `<Ausdruck>` wirkt wie Iterator

Beispiel: `int[] args; /* … */ for (int a : args) ;`

FOREACH-SCHLEIFE (2/2) – SEMANTIK



WHILE-SCHLEIFE, DO...WHILE-SCHLEIFE

```
<while-Schleife> = "while" '(' <logischer Ausdruck> ')'
                        ( <Anweisung> | <Block> )

<do…while-Schleife> = "do"
                         ( <Anweisung> | <Block> )
                      "while" '(' <logischer Ausdruck> ')' ';'
```

MEHRFACHVERZWEIGUNG

```
<switch-Anweisung> = "switch" '('
                        <ganzzahliger Ausdruck> |
                        <String-Ausdruck>
                     ')' '{'
                        { "case" (
                              <ganzzahlige Konstante> | <String>
                           ) ':'
                           [ ( <Anweisung> | <Block> ) ]
                           [ "break;" ]
                        }+
                        [ "default:"
                           [ ( <Anweisung> | <Block> ) ]
                        ]
                     '}'
```

Seit Java 7 wird String für die switch-Struktur unterstützt.

CONTINUE UND BREAK

`continue`

- nur für `for`-, `while`- und `do…while`-Schleifen gültig
- aktueller Schleifendurchlauf wird abgebrochen und es wird zum Schleifentest gesprungen
- nur bei `for`-Schleife: Reinitialisierung vor Schleifentest

`break`

- nur für `for`-, `while`- und `do…while`-Schleifen sowie `switch`
- nur die unmittelbar umgebende Kontrollstruktur wird verlassen

Hinweis: `continue` und `break` mit Sprungmarken sind erlaubt

GOTO, RETURN UND EXIT

`goto` Schlüsselwort, aber nicht erlaubt!

`return` Rückkehr in unmittelbar aufrufende Methode

Syntax: `'return' [ <Ausdruck> ] ';'`

`exit` liefert Rückkehrwert an das Betriebssystem (Standardmethode)

Syntax: `'System' '.' 'exit' <ganzzahliger Ausdruck> ';'`

Achtung! `return` in `foreach`-Schleifen hat Effekt wie `continue` auf vorheriger Folie!

LITERATUR

Java 2 SDK v 1.2.2, Grundlagen Programmierung, HERDT-Verlag für Bildungsmedien GmbH, Nackenheim

Guido Krüger: Handbuch der Java-Programmierung, Addison-Wesley Verlag, ISBN 3-8273-2201-4

Brit Schröter: KompaktReferenz Java2, DATA BECKER GmbH & Co. KG, ISBN 3-8158-1477-4

Internet HTML-Seiten, Wikipedia, …