Java7: Neue Sprachmerkmale und API's

W3L AG
info@W3L.de

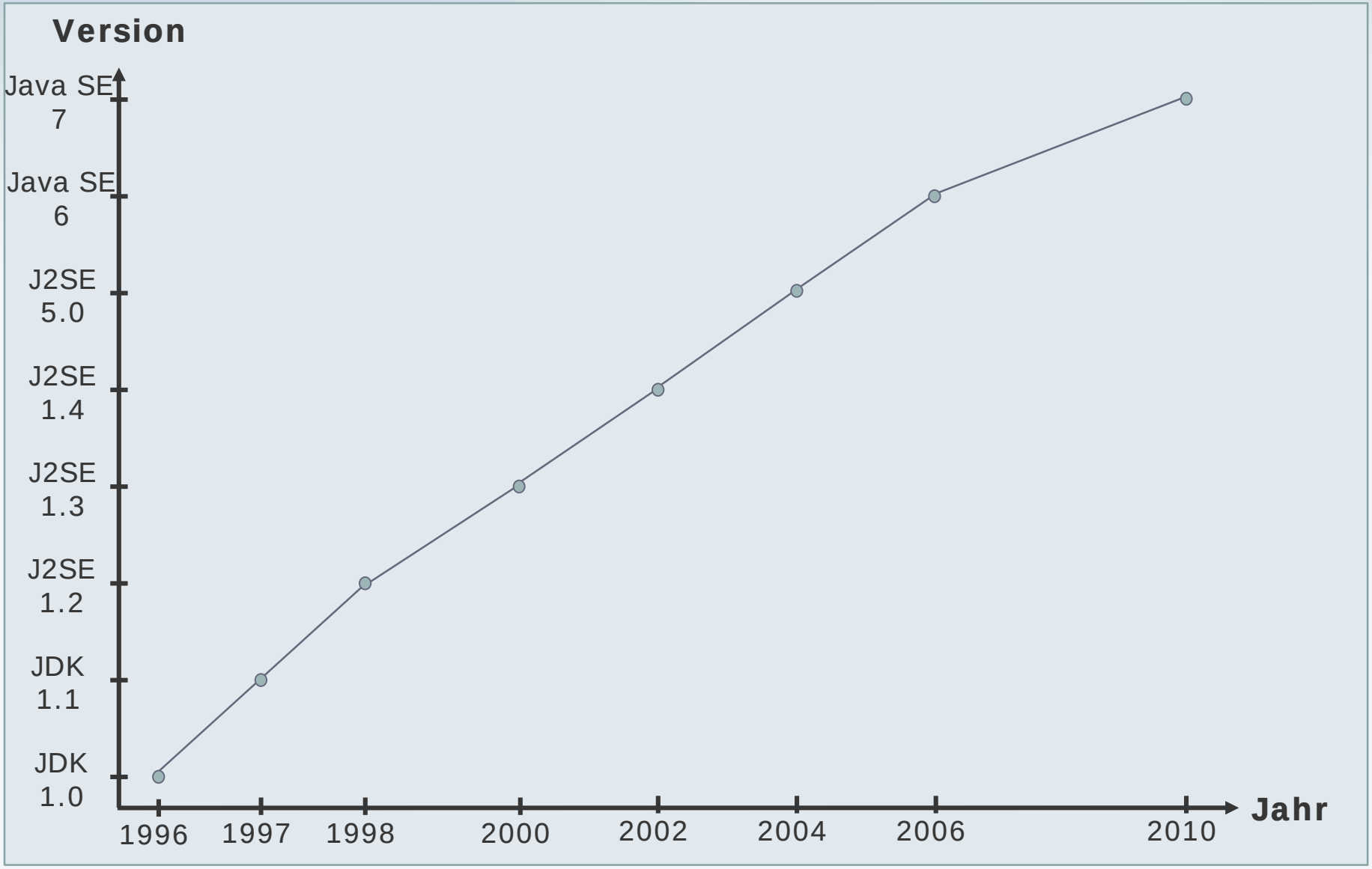
2009

Inhalt

- **Historische Entwicklung von Java**
  - Java Community Process und Java Specification Requests
  - JDK vs. OpenJDK
- **Was gibt es Neues?**
  - Sprachemerkmale
  - APIs
- **Fazit**

zurück

Historische Entwicklung von Java



Historische Entwicklung von Java

- **JDK 1.0 (1996)**
  - Initiales Release von Java
  - Standardpakete: `java.lang;java.io;java.util;java.net;java.awt;java.applet`
  - Geschichte
    - Unter dem Namen **The Green Project** im Auftrag von Sun zwischen 1991 und 1992 entwickelt.
    - Codename **Oak** (Eiche) / **O**bject **A**pplication **K**ernel
    - Entwicklungsziel: Programmiersprache, Betriebssystemumgebung und virtuelle CPU für unterschiedliche Einsatzzwecke!
    - Im Sommer 1992 erste Demonstration einer Oak-basierten Anwendung im Bereich digitaler Mehrwertdienste für das Kabelfernsehen.
    - Integration der Java Runtime in den frühen graphischen Webbrowser Mosaic. Später wurde daraus der HotJava-Browser.
    - Durchdruch gelingt aber erst durch Kooperation mit Netscape.
    - Im März 1995 wurde der Java-Quellcode freigegeben.
    - Im Mai 1995 wurde Java erstmals der Öffentlichkeit vorgestellt.

Historische Entwicklung von Java

- **JDK 1.1 (1997)**
  - AWT überarbeitet. Einführung des Ereignis-Delegationsmodells.
    - *Observer Pattern* mit Ereignisquellen und –senken!
  - U.a. deshalb Java-Syntax um innere Klassen erweitert.
  - Weitere Neuerungen
    - **RMI**
    - Java-Komponentenmodell und –plattform (**JavaBeans**)
    - JAR-Dateiformat
    - Reflection, wobei nur Introspektion und keine Metaprogrammierung
    - Zugriff auf relationale Datenbanken (**JDBC**)
    - Serialisierung von Objektstrukturen
    - Sicherheitsfunktionen
    - Ab Version 1.1.5 wurde **Swing** ausgeliefert. War jedoch nicht offizieller Teil des Java API.

Historische Entwicklung von Java

- **J2SE 1.2 (1998)**
  - Codename Playground.
  - Namensgebung wurde durch Sun geändert.
    - Sun spricht offiziell von Java 2!
    - Aus JDK wird J2SDK. Unterscheidung zwischen **J2EE** und **J2SE**.
  - **Just-In-Time-Compilation** zur Beschleunigung der Ausführungsgeschwindigkeit von Java-Programmen.
  - Der **Java Community Process** (JCP) wird ebenfalls eingeführt!
  - Weitere Neuerungen
    - Schnittstellen und Implementierungen für Collections
    - Swing und Java 2D
    - Java IDL für CORBA
    - **strictfp**-Schlüsselwort
    - Schwache Referenzen implementiert im Paket `java.lang.ref.Reference`.

Historische Entwicklung von Java

- **J2SE 1.3 (2000)**
  - Codename Kestrel.
  - **Hotspot JVM**
    - Hotspots werden automatisch erkannt und Just-In-Time in schnellen Maschinencode compiliert.
    - Closed-World-Optimierungen führen zu erheblichen Geschwindigkeitssteigerungen.
  - Java Sound API
  - Java Naming and Directory Interface (JNDI)
  - Java Platform Debugger Architecture (JPDA)
  - Erweiterungen an Java RMI. Kompatibilität zu CORBA.

Historische Entwicklung von Java

- **J2SE 1.4 (2002)**
  - Codename Merlin. Weiterentwicklung erfolgt innerhalb des Java Community Process unter **JSR 59**.
  - Erste Spracherweiterung nach JDK 1.1 für die Definition von **Assertions**.
  - API Ergänzungen für die Serverprogrammierung (z.B. Servlet API)
  - Java Web Start (JSR 56)
  - Weitere Neuerungen
    - Integrierter XML-Parser und XSLT-Prozessor (JSR 5 und JSR 63)
    - Java Authentication and Authorization Service (**JAAS**)
    - Java Secure Socket Extension (**JSSE**)
    - **NIO** (New Input/Output) für die Behandlung von großen IO-Mengen (JSR 51)

Historische Entwicklung von Java

- **J2SE 5.0 (2004)**
  - Codename Tiger. Spezifiziert durch JSR 176.
  - Wieder neue Java-Sprachemerkmale
    - Generics (JSR 14)
    - Annotationen (JSR 175)
    - Auto-Boxing und Auto-Unboxing (JSR 201)
    - Enumerations
    - Variable Argumente einer Operation
    - For Each-Schleifen
    - Statische Importe
  - Java Speicher Modell (JSR 133)
  - Weitere Neuerungen
    - Verbesserte Unterstützung der nebenläufigen Programmierung. Neues Paket `java.util.concurrent`.
    - Automatische Stub-Generierung für RMI-Objekte

Historische Entwicklung von Java

- **Java SE 6 (2006)**
  - Codename Mustang. Spezifiziert durch JSR 270.
  - Namensgebung wurde durch Sun geändert.
    - Aus J2SE wird Java SE! Kein ".0" hinter der führenden Versionnummer!
  - Wichtige Erweiterungen bzgl. Diagnose, Überwachung und Management.
  - Scripting Language Support (JSR 223).
  - Erhebliche Performanz-Erhöhung.
  - Ab Version 1.6.0_10
    - Next Generation Java Plugin. Applets werden in einem separaten Prozess ausgeführt.
    - Java Quick Starter. Initialisierungsphase der JVM wird verkürzt.
    - Verbesserte Java2D Performanz unter Windows.
  - Weitere Neuerungen
    - Verbesserte Webservice-Benutzung. JAX-WS (JSR 224)
    - JDBC 4.0 Unterstützung (JSR 221)
    - Java Compiler API (JSR 199)

Java Community Process

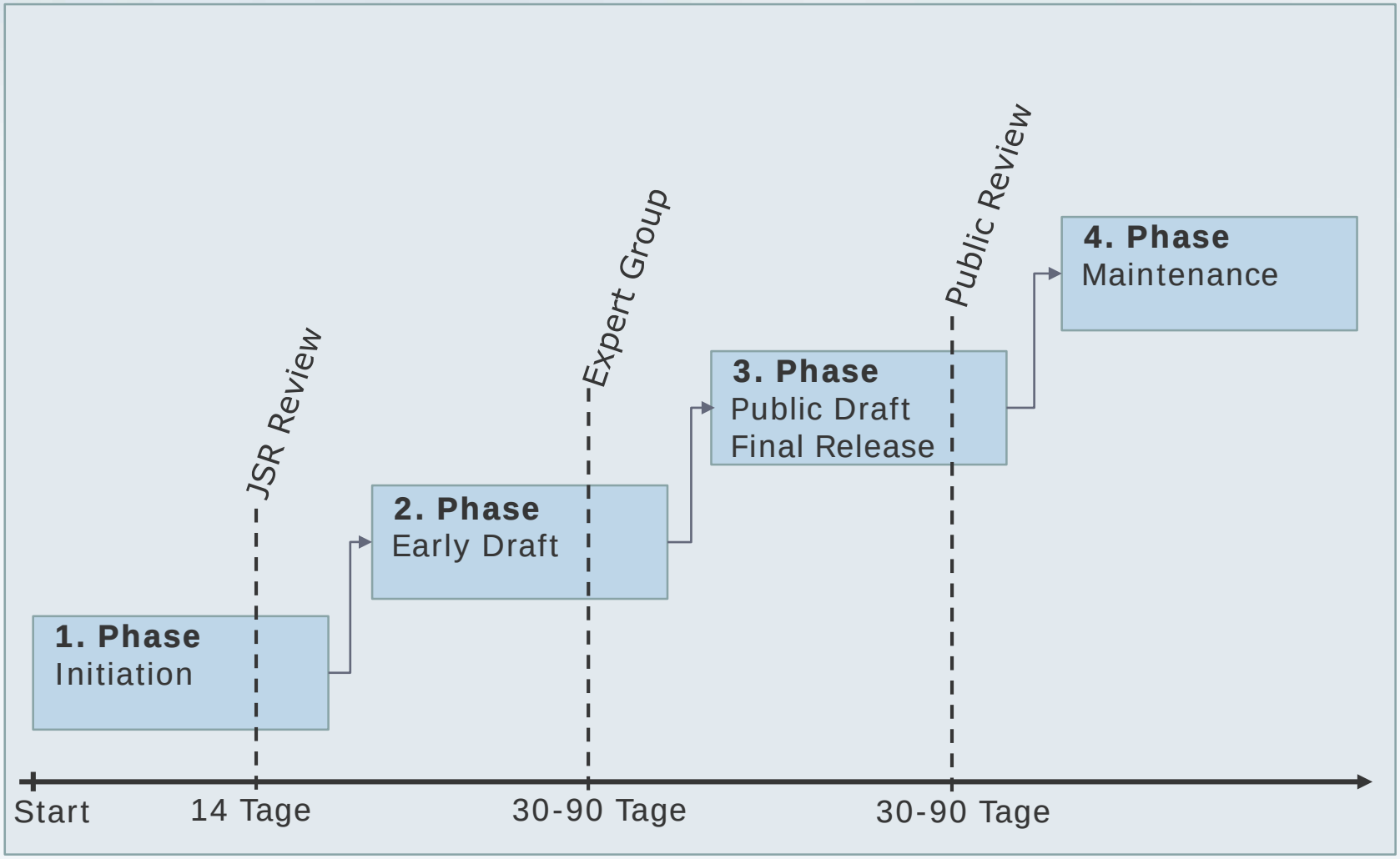
- **Ziele**
  - Seit ihrer Einführung im Jahr 1998 soll der *Java Community Process* der internationalen Entwicklergemeinde erlauben, an der Weiterentwicklung von Java teilzuhaben.
  - Es wird ein klarer Prozess definiert, in dessen Rahmen Änderungen an der *Java Technology Specification* durchgeführt werden können.
  - Die Firma Sun und JCP-Mitglieder agieren »der Theorie nach« auf Augenhöhe.
  - Kosten einer Mitgliedschaft
    - $5000/Jahr für die Industrie.
    - $2000/Jahr für Universitäten und Non-Profit-Organisationen.
    - $0/Jahr für Einzelpersonen oder Java-Lizenznehmer.

Java Community Process

- **Details**
  - Weiterentwicklungen oder Korrekturen an bestehenden Funktionen werden durch einen **Java Specification Request** initiiert.
    - Auch der vollständige Versionswechsel von Java läuft beispielsweise unter solchen JSR‘s. Beispiel J2SE 1.4 als JSR 59 oder J2SE 5.0 als JSR 176.
  - Der gesamte Prozess ist öffentlich.
    - Ohne JCP-Mitgliedschaft ist daher eine Einsicht in die Spezifikation möglich und das Kommentieren erwünscht.
  - JSRs werden von **Experten-Gruppen** *(expert groups)* entwickelt.
  - Ein **Vorstand** *(executive committee)* koordiniert und überwacht die Aktivitäten der Experten-Gruppen.
  - Das ***Program Management Office*** *(PMO)* überwacht den Java Community Process.

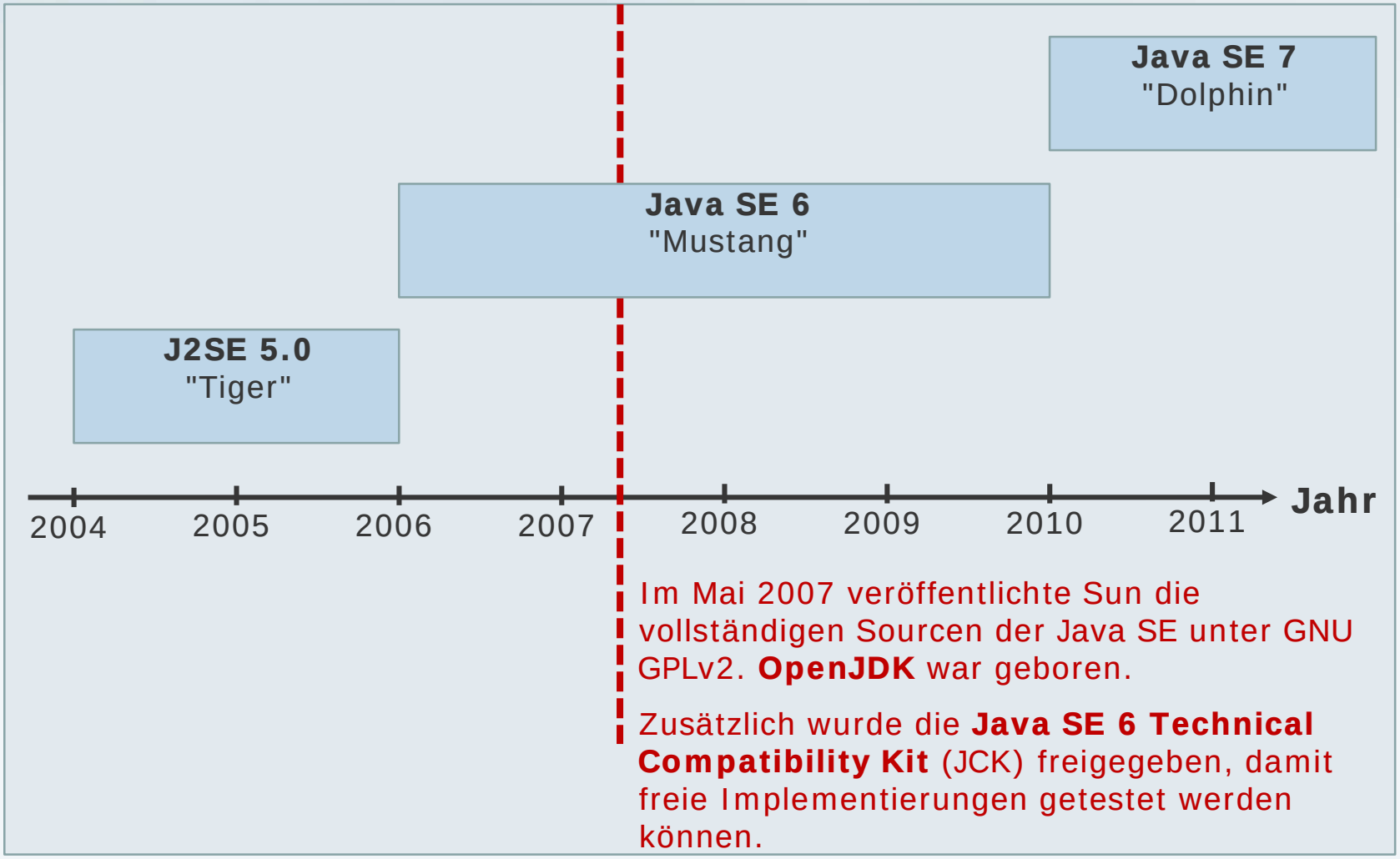
Java Community Process

Java Specification Request Zeitrahmen



JDK vs. OpenJDK

Zeitachse



Im Mai 2007 veröffentlichte Sun die vollständigen Sourcen der Java SE unter GNU GPLv2. **OpenJDK** war geboren.

Zusätzlich wurde die **Java SE 6 Technical Compatibility Kit** (JCK) freigegeben, damit freie Implementierungen getestet werden können.

JDK vs. OpenJDK

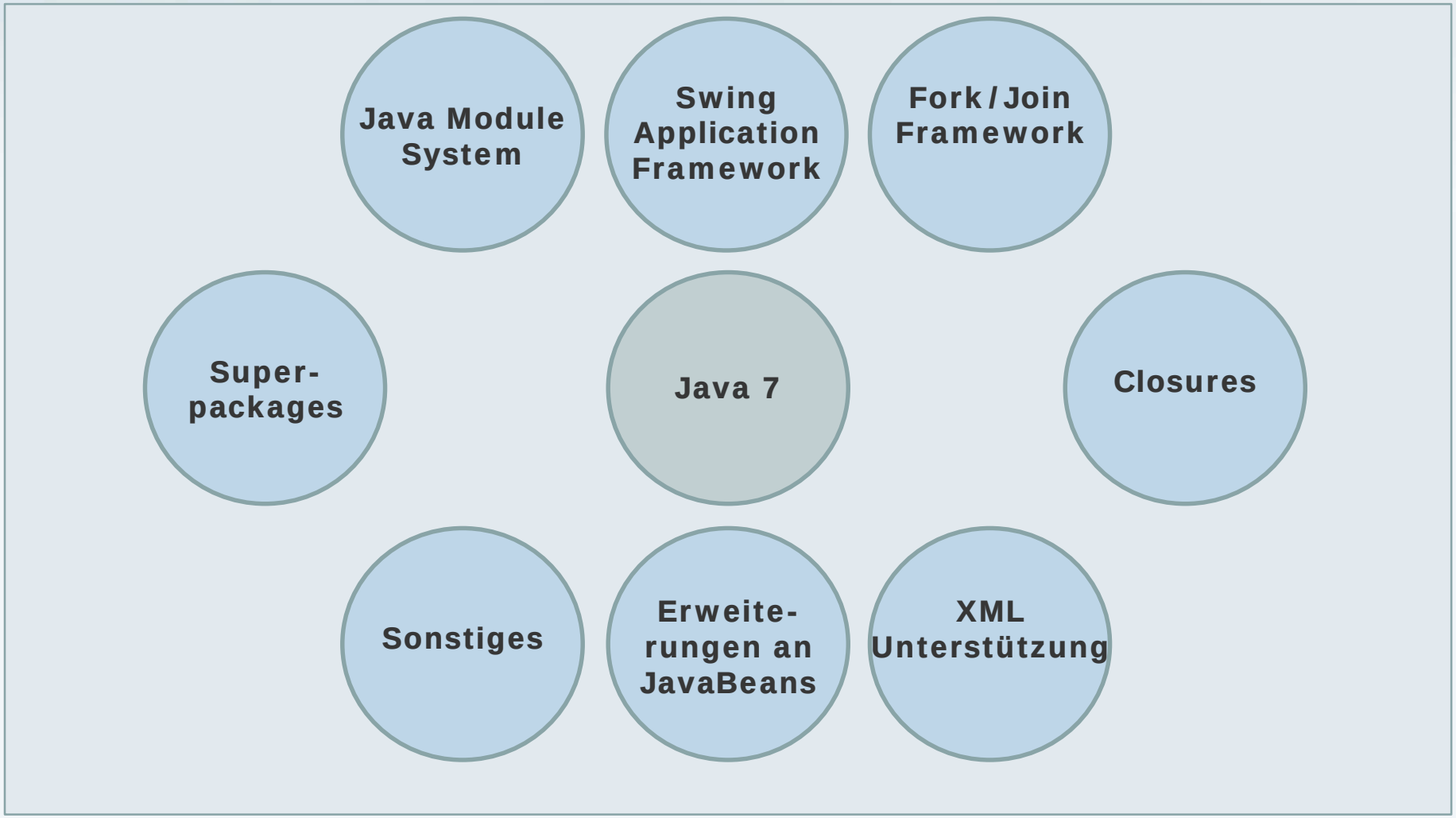
- **Begriffsdschungel**
  - **Java SE**, **JCK**, **JDK**, **OpenJDK**, **IcedTea**, **Harmony**?
- **Java SE**
  - Java Standard Edition ist nur noch eine Spezifikation!
- **Sun gesteuerte Implementierungen**
  - JDK7 (`http://jdk7.dev.java.net`)
    - Implementierung wird fast ausschließlich durch Sun durchgeführt. Interne Tools sind zum Release-Build der Software notwendig.
  - OpenJDK7 (`http://openjdk.java.net`)
    - Die internationale Entwicklergemeinde arbeitet an einer vollständigen Open-Source-Variante des JDK.
    - IcedTea ist ein Baustein des OpenJDK, welcher von Red Hat im Juni 2007 initiiert wurde. Das Ziel ist die Nach-Implementierung der nicht freien Software, auf die das JDK7-Projekt angewiesen ist.
- **Apache Harmony**
  - Von der Apache Foundation angestoßene Entwicklung der Java SE.
  - Implementierung des J2SE 5.0.

Was gibt es Neues?

- **Java SE 7 (Mitte 2010)**
  - Codename Dolphin. Spezifikation läuft nicht mehr unter einem JSR!
    - Es ist ein »Mischmasch« zwischen OpenJDK-Projekten und JCP geleiteten JSR‘s.
    - Was Java SE 7 ausmacht ist nicht mehr öffentlich spezifiziert! Der Java Community Process ist damit ausgehebelt worden!
    - Beispiel: Probleme der Apache Foundation mit dem J2SE 5 JCK für Apache Harmony!
  - JVM Unterstützung für dynamische Sprachen.
  - Neues Framework für die nebenläufige Programmierung auf Multi-Core-Prozessoren.
  - Geringe Spracherweiterungen. **Project Coin** OpenJDK
    - Strings in `switch`-Statements
  - Weitere Neuerungen
    - Superpackages
    - Swing Application Framework
    - Unterstützung für XML auf Sprachebene

Was gibt es Neues?

- Überblick



Was gibt es Neues?

- **Superpackages**
  - Spezifiziert in JSR 294.
  - Unterstützung des Geheimnisprinzips durch ein Java-Sprachmerkmal.
  - Verwandt mit JSR 277 (Java Module System) und JSR 291 (OSGi).
  - Warum?
    - Siehe Parnas: Kapselung und Verbergung von Implementierungsdetails führt zu besserer Software!
  - Bisherige Ansätze
    - Spezielle Classloader
    - Innere oder private Klassen
    - Fehlende Dokumentation

Was gibt es Neues?

- **Superpackages**
  - Beispiel

    Superpackages werden in Java-Source-Dateien definiert (Endung .java)

```
superpackage standard_java {
   member package java.util;
   member package java.io;
   member package sun.io;  //Implementierungsdetail

   //Öffentliches API des Superpackages!
   export java.util.*;
   export java.io.*;
}
```

Was gibt es Neues?

- **Superpackages**
  - Compile-Time vs. Run-Time
    - Superpackage-Definitionen werden durch den Java-Compiler in eine **.spkg**-Datei umgewandelt.
    - Eine Klasse definiert zwar zu welchem Paket, aber nicht zu welchem Superpackage sie gehört!
    - Diese Metainformationen stehen nur zur Laufzeit zur Verfügung.

Was gibt es Neues?

- **Java Module System**
  - Spezifiziert in JSR 277.
  - JAR's mit Metadaten
    - SuperJAR‘s bzw. JAva Module (JAM)
  - Möglichkeiten der Versionierung von Implementierungen.
  - Definition von Abhängigkeiten.
  - Ähnlichkeiten zu OSGi!

Was gibt es Neues?

- **Swing Application Framework**
  - Spezifiziert in JSR 296.
  - Ziel ist, die Entwicklung von Swing-Anwendungen zu erleichtern, indem eine kleine Menge von Infrastruktur-Klassen zur Verfügung gestellt wird.
  - Im Detail behandelt das Swing Application Framework
    - Lebenszyklus von Anwendungen (Startup und Shutdown)
    - Unterstützung der Verwaltung von Ressourcen.
    - Unterstützung bei der Definition von Aktionen. Undo/Redo-Funktionen und asynchrone Ausführung im Hintergrund.
    - Speicherung des Session-Zustands. U.a. der Größe des Anwendungsfensters.
  - Wichtige Klassen sind:
    - `ApplicationContext`, `Application`, `SingleFrameApplication`, `ActionManager`, `Session`, `Tasks`

Was gibt es Neues?

- **Swing Application Framework**
  - Beispiel A

```
public class MeineAnwendung extends Application {
  protected void startup(String[] args) {
       Jframe f = new Jframe("Meine Anwendung");
       f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
               exit(e);
            });
       f.setVisible(true);
  }
  protected void shutdown() {
  }
  public static void main(String[] args) {
     launch(MeineAnwendung.class, args);
  }
}
```

Was gibt es Neues?

- **Swing Application Framework**
  - Beispiel B

```
public class MeineAnwendung extends SingleFrameApplication {
  protected void startup(String[] args) {
     JLabel l = new JLabel("Mein Label");
     show(l);
}
public static void main(String[] args) {
     launch(MeineAnwendung.class, args);
  }
}
```

Was gibt es Neues?

- **Fork-Join-Framework**
  - Klassen zur abstrakteren Parallelisierung von Algorithmen.
  - Grund
    - Die Takterhöhung stößt an physikalische Grenzen.
    - Moderne CPU-Architekturen besitzen mehr als einen Kern.
    - Echt-parallele Abarbeitung von Anweisungen verschiedener Threads ist daher möglich.
    - Software-technische Konzentration auf diesen Sachverhalt »hinkt« der Entwicklung hinterher.
  - Eignet sich hervorragend, um **devide-and-conquer-Algorithmen** zu parallelisieren.

Was gibt es Neues?

- **Fork-Join-Framework**
  - Beispiel A

```
//Pseudocode
Ergebnis berechne(Problem p)
{
   Ergebnis links, rechts;

   INVOKE-PARALLEL { //Fork
       links = berechne(teileLinks(p));
       rechts = berechne(teileRechts(p));
   } //Join

   return kombiniere(links, rechts)
}
```

Was gibt es Neues?

- **Fork-Join-Framework**
  - Beispiel B

```
public class SelectMaxProblem
{
   private final int[] feld;
   private final int start;
   private final int end;
   //Konstruktor wurde weggelassen…
   public int size() {return end – start;}
   public int sucheMaxSequentiell() {
      int max = Integer.MIN_VALUE;
      for (int i = start; i < end, i++) {
          if (numbers[i] > max)
              max = numbers[i];
      }
      return max;
   }
   public SelectMaxProblem teile(int s, int e) {
      return new SelectMaxProblem(feld, start + s, end + e);
   }
}
```

Was gibt es Neues?

- **Fork-Join-Framework**
  - Beispiel B

```
public class SucheMaxFJ extends RecursiveAction
{
  private final int schwelle;
  private final SelectMaxProblem p;
  public int ergebnis;
  //Konstruktor wurde weggelassen…
  protected void compute() {
     if (p.size() < schwelle)
       ergebnis = p.sucheMaxSequentiell();
     else {
       int mitte = p.size() / 2;
       SucheMaxFJ l=new SucheMaxFJ(p.teile(0, mitte), schwelle);
       SucheMaxFJ r=new SucheMaxFJ(p.teile(mitte+1,p.size()), schwelle);
       coInvoke(l, r);
       result = Math.max(l.ergebnis, r.ergebnis);
     }
  }
}
```

Was gibt es Neues?

- **Fork-Join-Framework**
  - Beispiel B

```
public class SucheMaxFJ extends RecursiveAction
{
  public static void main(String[] args) {
     SelectMaxProblem p = new SelectMaxProblem(feld, 0, laenge);
     int schwelle = 100;
     int threads = 16;
     SucheMaxFJ max = new SucheMaxFJ(p, schwelle);

     ForkJoinExecutor pool = new ForkJoinPool(threads);
     pool.invoke(max);

     System.out.println(max.result);
  }
}
```

(vgl. *„Java theory and practice: Stick a fork in it“*, Brian Goetz)

Was gibt es Neues?

- **Java-Closures**
  - Es ist bis jetzt nicht klar, ob sie in Java 7 realisiert werden. Eher nicht!
  - Fehlendes Konzept der Funktionszeiger (vgl. Delegates).
  - Anonyme innere Klassen erfordern zu viel Schreibarbeit!
  - Beispiel

```
public void meineOperation() {
    (new Thread(new Runnable() {
        public void run() {
            System.out.println("Hallo Welt!");
        }
     })).start();
}
```

  - Deklaration eines Closures

```
{Parameter => Expression;}
```

  - Beispiel

```
void opName() {System.out.println("Hallo Welt!");} //Normale Deklaration
{ => System.out.println("Hallo Welt!");} //Closure Deklaration
```

Was gibt es Neues?

- **Java-Closures**
  - Ein Java-Closure steht für ein Objekt, welches eine `invoke()`-Operation besitzt.
  - Beispiel

```
public static void main(String[] args)
{
  //Aufruf eines Closures ohne Parameter!
  {=> System.out.println("Hallo Welt!");}.invoke();

  //Aufruf eines Closures mit Parameter!
  {String s => System.out.println("Hallo Welt!" + s);}.invoke("test");

}
```

  - Referenzen auf Closures

```
{ => void} c1 = {=> System.out.println("Hallo");};
{String => void} c2 = {String s => System.out.println("Hallo");};
```

  - Als Parameter

```
public void operation({ => void} delegate);
```

Was gibt es Neues?

- **Java-Closures**
  - Closure Conversion
    - Ein Closure lässt sich einer Schnittstelle zuweisen, sofern Rückgabe- und Parametertypen übereinstimmen.

```
Runnable test = { => System.out.println("Test");};
```

  - Beispiel

```
public void meineOperation() {
    (new Thread({=> System.out.println("Hallo Welt!");})).start();
}
```

Was gibt es Neues?

- **XML-Unterstützung auf Sprachebene**
  - Support für XML Datenkonvertierung, Navigation, Streaming und Sprachsupport

```
public void addChild(XML node, String text)
{
   node.add(<textnode>{text}</textmode>);
}
```

Was gibt es Neues?

- **JavaBeans**
  - Binding von JavaBean-Properties aneinander
    - Spezifiziert in JSR 295.
    - Automatischer Synchronisationsmechanismus.
    - Definition per JSP-Expression Language.
    - Anwendungsbeispiel: Swing-Properties mit Backend-Properties verbinden!
  - Beans-Validation-Framework
    - Spezifiziert in JSR 303.
    - Constraints auf Beans via Annotations (vgl. Hibernate).
    - Beispiel: `@NotNull`
  - Keywords für Properties

```
//Readonly Property
final property String name=„Max Mustermann";
//Read/Write Property
property String nummer;
```

Was gibt es Neues?

- **Annotationen**
  - Spezifiziert in JSR 308.
  - Annotationen lassen sich an weiteren Stellen verwenden.
    - Typ-Deklarationen, Variablen-Definition, Ausdrücke, Typecasts, usw.
  - Mehr Standard-Annotationen
    - Spezifiziert in JSR 305.
    - Beispiele

      `@NonNull`, `@ReadOnly`, `@UnmodifiableList`, `@NonEmpty`

Was gibt es Neues?

- **Skriptsprachen**
  - Spezifiziert in JSR 292.
  - Mehr Skriptsprachen im JDK.
  - Um eine hohe Performanz zu erzielen, sollen neue Funktionen auf der Ebene der virtuellen Maschine hinzugefügt werden.
  - Invokedynamic im Java-Bytecode
    - Das Ziel eines Methodenaufrufs kann dynamisch bestimmt werden.

Was gibt es Neues?

- **Generics**
  - Laufzeit-Unterstützung für Generics.
  - Informationen stehen vor Java 7 nur zur Compilierungszeit zur Verfügung.
  - Erweiterungen an `java.lang.reflect.*` für die Laufzeit.

Was gibt es Neues?

- **NIO 2**
  - Spezifiziert in JSR 203.
  - Vervollständigung von NIO, welches mit Java 5 eingeführt worden ist.
    - Asynchrone APIs für den Zugriff auf Sockets und Dateien.
    - Ereignisse bei Änderungen im Dateisystem.
    - Neues API, um u.a. auf Dateisystemattribute zuzugreifen.
    - Multicast Unterstützung.
    - Buffergrößen jenseits der `int`-Grenzen!

Was gibt es Neues?

- **Weitere API-Erweiterungen**
  - JMX 2.0
    - Spezifiziert in JSR 255.
  - JMX Remote Connector für Webservices
    - Spezifiziert in JSR 262.
  - Uhrzeiten
    - Spezifiziert in JSR 310.
    - JodaTime-Bibliothek dient als Vorlage der neuen API.
    - ISO 8601
  - Einheiten
    - Spezifiziert in JSR 275.
    - Neue Klassen für den Umgang mit Einheiten und Quantitäten.
    - `KILO(METER).getConverterTo(MILE).convert(10);`

Was gibt es Neues?

- **Weitere Sprachmerkmale (Projekt *Coin*)**
  - switch-Statement
    - Erweiterung um String-Literale

```
switch(subject) {
   case "yes": break;
   case "no": break;
}
```

  - Mehrere Exceptions in einer Anweisung abfangen

```
catch(IOException, NullPointerException e) {}
```

  - Aufzählungstypen
    - Vergleichoperatoren bei Enumerationen.
    - Definierbare Werte bei Aufzählungen.

Fazit

- **Java Weiterentwicklungsprozess ist undurchsichtiger geworden!**
- **Mitspracherecht der OpenSource-Gemeinde besteht nur auf dem Papier!**
- **Gefahr von Inkompatibilitäten verschiedener Java-JDKs!**
  - Sorgfältige Auswahl des JDK.
  - Wechsel zu einer anderen Implementierung muss wohl überlegt sein.
- **Interessante Neuerungen**
  - Superpackages
  - Unterstützung der nebenläufigen Programmierung
  - JavaBean-Properties à la .NET
  - Closures

Inhouse-Schulungen

Wir bieten Inhouse-Schulungen und Beratung durch unsere IT-Experten und –Berater.

Schulungsthemen

- Softwarearchitektur (OOD)
- Requirements Engineering (OOA)
- Nebenläufige & verteilte Programmierung

Gerne konzipieren wir auch eine individuelle Schulung zu Ihren Fragestellungen.

Sprechen Sie uns an!
Tel. 0231/61 804-0, info@W3L.de

W3L-Akademie

In Zusammenarbeit mit der Fachhochschule Dortmund bieten wir

zwei Online-Studiengänge

- B.Sc. Web- und Medieninformatik
- B.Sc. Wirtschaftsinformatik

und 7 Weiterbildungen im IT-Bereich an.

Besuchen Sie unsere Akademie!
http://Akademie.W3L.de