

# Neues im JDK 1.5

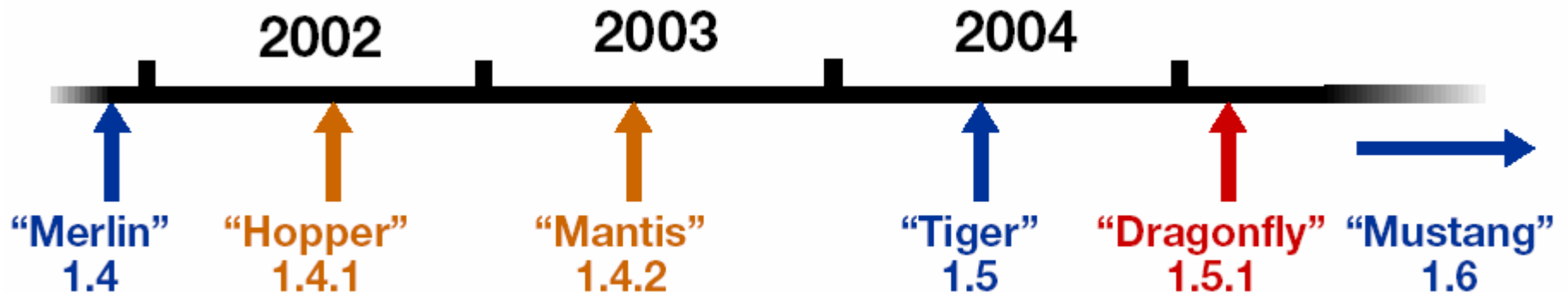
## Praktikum aus Softwareentwicklung 2

# J2SE 1.5.0 Tiger

- New Features
  - Timeline, Überblick
- Java Spracherweiterungen
  - Enums, For Loop, ..
- Java API
  - StringBuilder, Formatter
- Sonstiges



# J2SE Plattform Versionen Timeline

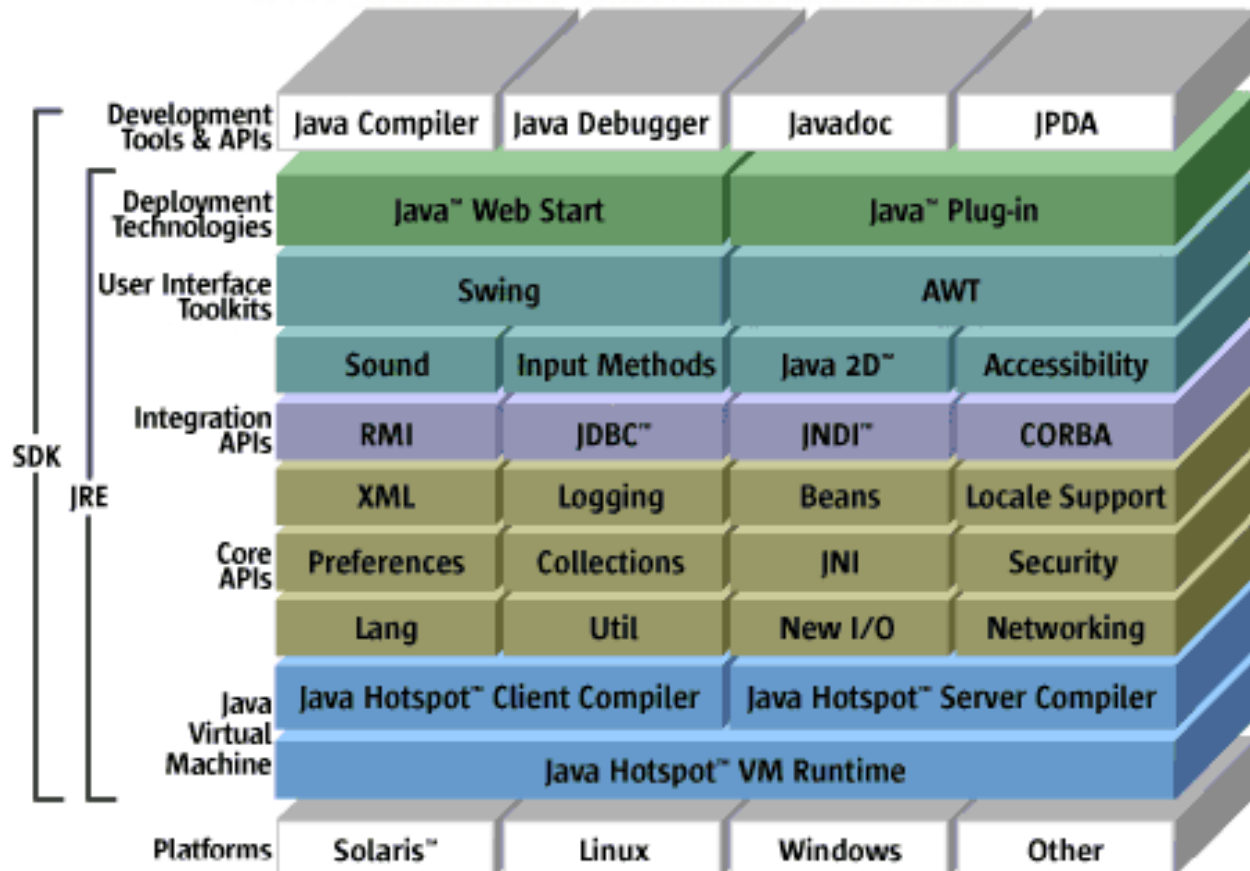


Aktuelle Versionen:

- Jdk 1.4.2\_12
- Jdk 1.5.0\_09
- Jdk 6 Beta 2

# Jsdk1.5 Übersicht

## Java™ 2 Platform, Standard Edition v 1.5



# J2SE 1.5 New Features 1

## Java Virtual Machine

- Java Heap Self Tuning
  - (Selbst)-Einstellung des Heap wurde optimiert (Performance und Speicherverbrauch)
- Class Data Sharing
  - Soll Startzeit bei kleinen Applikationen reduzieren
  - Teile der Standardbibliotheken werden in ein Memory-Mapped-File ausgelagert
  - `jre\bin\client\classes.jsa`
  - MMF wird zwischen mehreren VM's geteilt
- Garbage Collector Ergonomics
  - Parallele Garbage Collector verbessert
  - Big Size Verbesserungen, besseres adaptives Verhalten
- Fatal Error Handling

# J2SE 1.5 New Features 2

## Java Spracherweiterungen 1

- Autoboxing/Unboxing
  - Abschaffung der manuellen Konvertierung von primitiven Typen und ihren Wrapper Klassen
- Enhanced For Loop
  - Neues Schleifen Konstrukt für Collections und Arrays
- Generics
  - Bieten Typsicherheit zur Kompilierzeit für Collections und sorgen dafür, dass nicht bei jeder Entnahme eines Objekts aus den Collections ein Casting erforderlich wird
- Metadata
  - „Annotations“, ermöglicht deklarative Programmierung
  - Kein Schreiben von wiederkehrenden Informationen mehr erforderlich → wird von Tools erledigt

# J2SE 1.5 New Features 3

## Java Spracherweiterungen 2

- **Static Import**
  - Bei Verwendung von statischen Member-Variablen anderer Klassen müssen keine Klassennamen mehr benutzt werden. Dadurch wird der Code übersichtlicher
- **Typesafe Enums**
  - Liefern alle Vorteile des Typesafe enum-Pattern
- **Varargs**
  - Variable Argumentlisten bei Methoden Deklaration
  - Variable Argumentlisten brauchen nicht mehr in Arrays oder Collections gespeichert werden um sie einer Methode zu übergeben

# J2SE 1.5 New Features 4

## Core Libraries (1/2)

- Network
  - Timeout (Protocol Handler), InetAddress ping, Framework für File Caching
- Internationalisierung
  - Unicode 4.0, Vietnam Locale
- Formatter (java.util)
  - Interpreter Klasse für printf-Style Format Strings

```
String s = String.format("Duke's Birthday: %1$tm
                                %1$te,%1$tY", calendar);
```

-> s == "Duke's Birthday: May 23, 1995"



# J2SE 1.5 New Features 5

## Core Libraries (2/2)

- **Scanner** (`java.util`)
  - Konvertiert Text in Primitives oder Strings
- **Collections**
  - 3 neue Interfaces (`Queue`, `BlockingQueue`, `ConcurrentMap`)
  - Einige neue Algorithmen
- **Monitoring, Managment, Instrumentation**
- **Concurrency Utilities**
- **Bit Manipulation Operations**
- <http://java.sun.com/j2se/1.5.0/docs/relnotes/features.html>

# J2SE 1.5.0 Tiger

- New Features
  - Timeline, Überblick
- Java Spracherweiterungen
  - Enums, For Loop, ..
- Java API
  - StringBuilder, Formatter
- Sonstiges

# Java Spracherweiterungen

## Typdafe Enumerations (1/4)

- Typsichere Enumerationen weisen die folgenden Eigenschaften auf:
  - Sie bieten Typsicherheit zur Kompilierzeit.
  - Sie sind Objekte und können daher in Collections integriert werden.
  - Sie sind als Klasse implementiert, so dass eigene Methoden hinzugefügt werden können.
  - Sie bieten einen ordnungsgemäßen Namensraum für den Typ Enumeration.
  - Ihre ausgegebenen Werte sind informativ – wenn man dagegen ein `int` enum ausgibt, sieht man nur eine Zahl, die unter Umständen nicht so aussagekräftig ist.

# Java Spracherweiterungen

## Typsafe Enumerations (Bsp 1) (2/4)

```
public class EnumTest1 {  
    public enum Color {red, green, blue};  
    public static void main(String args[]) {  
        System.out.println(java.util.Arrays.asList(Color.values()));  
        for (Color aColor : Color.values()) {  
            switch(aColor) {  
                case red:  
                    System.out.println("Got red.");  
                    break;  
                case green:  
                    System.out.println("Got green.");  
                    break;  
                case blue:  
                    System.out.println("Got blue.");  
                    break;  
            } } } }
```

# Java Spracherweiterungen

## Typsafe Enumerations (Bsp 2) (3/4)

```
public class EnumTest2 {  
    public enum Coin {  
        OneCent(1), FiveCent(5), TenCent(10), Quarter(25);  
        private final int value;  
  
        Coin(int value) {  
            this.value = value;  
        }  
  
        public int value() {  
            return value;  
        }  
    }  
}
```

# Java Spracherweiterungen

## Typesafe Enumerations (Bsp 2) (4/4)

```
public class CoinTest {
    public static void main(String[] args) {
        for (EnumTest2.Coin c : EnumTest2.Coin.values())
            System.out.println(c+": \t"+c.value()+" cent \t"+color(c));
    }
    private enum Coin2Color { copper, nickel, silver }
    private static Coin2Color color(EnumTest2.Coin c) {
        switch(c) {
            case OneCent: return Coin2Color.copper;
            case FiveCent: return Coin2Color.nickel;
            case TenCent:
            case Quarter: return Coin2Color.silver;
            default: throw new AssertionError("Unknown Coin: " + c);
        }
    }
}
```

# Java Spracherweiterungen

## Autoboxing/Unboxing (1/2)

- Eliminiert das „lästige“ Umwandeln von primitiven Datentypen in ihre Wrapper Klassen.
  - Integer  $\leftrightarrow$  int
  - Double  $\leftrightarrow$  double
  - Etc...
- Umwandeln übernimmt der Compiler
- Achtung Nullpointer Exceptions möglich

# Java Spracherweiterungen

## Autoboxing/Unboxing (2/2)

```
public static void autoBox14() {  
    int x = 44;  
    Integer y = Integer.valueOf(x);  
    int z = y.intValue();  
}
```

```
public static void autoBox15() {  
    int x = 44;  
    Integer y = x;  
    int z = y;  
}
```

```
Integer a = null;  
int x = a;  
System.out.println("a is: " + x); // NullPointerException
```



# Java Spracherweiterungen

## Generics (1/7)

- Beim *Casten* eines Objektes gibt man den Typ eines Objektes an. Man glaubt zu wissen, welches Objekt man hat
  - `String s = (String)object;`
- Warum kann das nicht der Compiler für uns machen? Wir sagen ihm welches Objekt wir erwarten und er macht das *Casten* für uns.
- → Typsicherheit zur Kompilierzeit und wahrscheinlich weniger `ClassCastException`s während der Laufzeit.
- Generics sind nicht Templates (C++)
- Nichts magisches, Compiler erledigt die Casts

# Java Spracherweiterungen

## Generics (Bsp 1.4) (2/7)

```
public static void printOS14() {  
    List osSystems = new ArrayList();  
    osSystems.add("Linux");  
    osSystems.add("Mac OS");  
    osSystems.add(new String("Solaris"));  
    osSystems.add(new StringBuffer("DOS"));    // runtime error  
    Iterator it = osSystems.iterator();  
    try {  
        while (it.hasNext()) {  
            String s = (String)it.next();  
            System.out.println("OS System has length " + s.length());  
        }  
    } catch (ClassCastException ce) {  
        ce.printStackTrace();  
    }  
}
```

# Java Spracherweiterungen

## Generics (Bsp 1.5) (3/7)

```
public static void printOS15() {  
    List<String> osSystems = new ArrayList<String>();  
    osSystems.add("Windows");  
    osSystems.add("Unix");  
    osSystems.add(new String("Free BSD"));  
    // compile error  
    // osSystems.add(new StringBuffer("Novell"));  
    Iterator<String> it = osSystems.iterator();  
    while (it.hasNext()) {  
        System.out.println("OS System has length " + it.next().length());  
    }  
}
```

# Java Spracherweiterungen

## Generics (Bsp 1.4) (4/7)

```
public static void list14() {
    try {
        List ys = new LinkedList();
        ys.add("zero");
        ys.add("one");
        List yss;
        yss = new LinkedList();
        yss.add(ys);
        String y = (String)((List)yss.iterator().next()).iterator().next();
        Integer z = (Integer)ys.iterator().next();
    } catch (ClassCastException ce) {
        ce.printStackTrace();
    }
}
```

# Java Spracherweiterungen

## Generics (Bsp 1.5) (5/7)

```
public static void list15() {  
    List<String> ys = new LinkedList<String>();  
    ys.add("zero");  
    ys.add("one");  
    List<List<String>> yss;  
    yss = new LinkedList<List<String>>();  
    yss.add(ys);  
    String y = yss.iterator().next().iterator().next();  
    // compile error  
    // Integer z = ys.iterator().next();  
}
```

# Java Spracherweiterungen

## Generics (Bsp 1.5) (6/7)

```
class Collections {  
    public static <S,T extends S> void copy(List<S> dest, List<T> src)  
        {...}  
}
```

```
class Collection<E> {  
    public <T> boolean containsAll(Collection<T> c) {...}  
    public <T extends E> boolean addAll(Collection<T> c) {...}  
}
```

# Java Spracherweiterungen

## Generics (Bsp 1.5) (7/7)

- Generics vs. Templates
  - Unlike C++, generic declarations are type checked
  - Generics are compiled once and for all
  - No code bloat
  - Generic source code not exposed to user - No hideous complexity
  - No template meta-programming
  - **Simply provide compile-time type safety and eliminate need for casts**

# Java Spracherweiterungen

## Enhanced For Loop (1/3)

- Iterationen über Collections sind aufwendig
- Der Iterator wird nur für das Holen des Elements verwendet
- Iteratoren sind fehleranfällig
  - Iterator-variable erscheint 3 mal pro Schleife
  - Common Cut and Paste errors
- Warum kümmert sich nicht der Compiler um den Iterator?



# Java Spracherweiterungen

## Enhanced For Loop (2/3)

```
try {  
    Iterator it = osSystems.iterator();  
    while (it.hasNext()) {  
        String s = (String)it.next();  
        System.out.println("OS System has length " + s.length());  
    }  
} catch (ClassCastException ce) {  
    ce.printStackTrace();  
}  
  
for (String s: osSystems) {  
    System.out.println("OS System has length " + s.length());  
}
```

# Java Spracherweiterungen

## Enhanced For Loop (3/3)

- Funktioniert auch für Arrays

```
int[] x = {10,20,30,40,50};  
for (int i : x) System.out.println(i);
```

- **Achtung:**

```
for (int i : x)
```

**Wird zu**

```
for (int g=0;g<x.length;++g) {  
    int i = x[g];  
    ...  
}
```

**Dieser Code funktioniert nicht!**

```
for (int i : x) .. if i == 55 ..;
```

# Java Spracherweiterungen

## Metadata (1/3)

- Das Metadaten-Feature ist darauf ausgelegt, den Entwicklern durch die Unterstützung der von Anbietern bereitgestellten Tools die Arbeit zu erleichtern
- Annotations
- Viele API's benötigen zweit Files (EJB's) → Tools sollen diese generieren
- Beispiel: JAX-RPC Webservice

# Java Spracherweiterungen

## Metadata (2/3)

```
public interface CoffeeOrderIF extends java.rmi.Remote {  
    public Coffee [] getPriceList() throws java.rmi.RemoteException;  
    public String orderCoffee(String name, int quantity) throws  
        java.rmi.RemoteException;  
}
```

```
public class CoffeeOrderImpl implements CoffeeOrderIF {  
    public Coffee [] getPriceList() {  
        ...  
    }  
  
    public String orderCoffee(String name, int quantity) {  
        ...  
    }  
}
```

# Java Spracherweiterungen

## Metadata (3/3)

```
// JAX-RPC Web Service with Metadata
import javax.xml.rpc.*;

public class CoffeeOrder {
    @Remote public Coffee [] getPriceList() {
        ...
    }

    @Remote public String orderCoffee(String name, int quantity) {
        ...
    }
}
```

# Java Spracherweiterungen

## Static Import (1/3)

- Konstanten werden oft in eigenen Klassen definiert:

```
public class Physics {  
    public static final double AVOGADROS_NUMBER = 6.02214199e23;  
    public static final double BOLTZMANN_CONSTANT = 1.3806503e-23;  
    public static final double ELECTRON_MASS = 9.10938188e-31;  
}
```
- Bei Verwendung muss Konstante qualifiziert werden:

```
double molecules = Physics.AVOGADROS_NUMBER * moles;
```
- Oft wird statt dessen ein „Konstanten-Interface“ definiert → **Antipattern**
- **Lösung: Static Import** Möglichkeit

# Java Spracherweiterungen

## Static Import (2/3)

- Analog zur Package Import Möglichkeit
- Importiert die statischen Mitglieder einer Klasse, anstatt die Klasse eines Packages
- Einzel Import oder alles
- Importiert auch Methoden und Enums
- Ist das wirklich ein Vorteil???
  - Viele IDE's bieten automatischen Import
  - Wird Code wirklich lesbarer?

# Java Spracherweiterungen

## Static Import (3/3)

```
import static org.iso.Physics.*;  
public class Guacamole {  
    public static void main(String[] args) {  
        double molecules = AVOGADROS_NUMBER * moles;  
        ...  
    }  
}
```

```
import static java.lang.Math.*;  
    // Statt: x = Math.cos(Math.PI * theta);  
x = cos(PI * theta);
```



# Java Spracherweiterungen

## Varargs (1/2)

- Um an eine Methode eine variable Anzahl von Parametern zu übergeben, muss man ein Array verwenden
- Arrays anlegen und füllen ist mühsam
- Funktionalität wie printf in C,C++ ist nicht möglich

// Bisher

```
public static void printem(String x[]) {  
    for (int t=0;t<x.length;++t) {  
        System.out.println(x[t]);  
    }  
}  
  
public static void main(String args[]) {  
    printem(new String[] {"bob","fred","joe"});  
}
```

# Java Spracherweiterungen

## Varargs (2/2)

// Neu

```
public static void printem(String... x) {  
    for (int t=0;t<x.length;++t) {  
        System.out.println(x[t]);  
    }  
}  
  
public static void main(String args[]) {  
    System.out.println("JDK1.5 Style!");  
    printem("bob", "fred", "joe");  
    System.out.println("JDK1.4 Style!");  
    printem(new String[] {"bob", "fred", "joe"});  
}
```

# J2SE 1.5.0 Tiger

- New Features
  - Timeline, Überblick
- Java Spracherweiterungen
  - Enums, For Loop, ..
- Java API
  - StringBuilder, Formatter
- Sonstiges

# Java API News

## StringBuilder

- Bisher war StringBuffer die bevorzugte Klasse um Strings zusammenzubauen.
  - Strings immutable
  - StringBuffer ist synchronisiert
- StringBuilder ist ein Plug-In Replacement für StringBuffer
- StringBuilder ist nicht synchronisiert
  - Bessere Performanz
  - `java.lang.StringBuilder`

# Java API News

## Stack Traces

- Neue Diagnose API's

```
public class StackTrace1 {  
  
    public static void main(String args[]) {  
        java.lang.StackTraceElement ste[] =  
            Thread.currentThread().getStackTrace();  
        for (int i=0; i <ste.length; i++) {  
            System.out.println(ste[i]);  
        }  
  
        System.out.println(Thread.getAllStackTraces());  
    }  
}
```

# Java API News

## Formatted Output/Input (1/4)

- Häufig verwendete Funktion in C: `sprintf(..)`;  
Anstatt mehrere Strings durch „+“  
Zusammenzuhängen, wird ein einzelner String mit  
dem Format und die Argumente an eine Methode  
übergeben.

```
String firstName = "Bill";  
String lastName = "Gates";  
String s = "Mein Name ist: " + firstName + " " + lastName;
```

```
String s = String.format("Mein Name ist: %1$s %2$s",  
                        firstName, lastName);
```

# Java API News

## Formatted Output/Input (2/4)

- Die `java.util.Formatter` class, ist die Ausgangsklasse.
- Hier sind alle `printf`-Optionen erklärt
- Bsp.: PI mit 10 Dezimalstellen

`java.lang.Appendable`

```
StringBuilder sb = new StringBuilder()  
Formatter formatter = new Formatter(sb);  
formatter.format("PI = %12.10f", Math.PI);  
System.out.println(formatter);
```

# Java API News

## Formatted Output/Input (3/4)

- Viele Klassen bieten die Methode `format` an zb. `PrintStream`:  
`System.out.format("Today is %1$tB %1$te, %1$tY.",  
Calendar.getInstance());`

→ Today is May 24, 2004.

`System.out.printf("Today is %1$tB %1$te, %1$tY.",  
Calendar.getInstance());`

→ Today is May 24, 2004.



# Java API News

## Formatted Output/Input (4/4)

- Mit Hilfe der Scanner Klasse können formatierte Strings von einem Stream gelesen werden:

```
Scanner s = new Scanner(System.in);  
String param= s.next();  
int value=s.nextInt();  
s.close();  
System.out.println("First Param was: " + param);  
System.out.println("Second Param was: " + value);
```

# Java API News

## Properties (1/2)

- Für Konfigurationszwecke gibt es bisher die Möglichkeit Key und Values in einem Property-File zu speichern:

```
key =value
```

```
key2 = value2
```

- Nun gibt es die Möglichkeit die Properties auch im XML Format abzuspeichern

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE properties SYSTEM
```

```
"http://java.sun.com/dtd/properties.dtd">
```

```
<properties>
```

```
  <comment>Hi</comment>
```

```
  <entry key="key">value</entry>
```

```
  <entry key="key2">value2</entry>
```

```
</properties>
```

# Java API News

## Properties (2/2)

```
Properties prop = new Properties();  
FileInputStream fis = new FileInputStream("props.xml");  
prop.loadFromXML(fis);  
prop.list(System.out);  
System.out.println("\nThe key property: " +  
                    prop.getProperty("key"));
```

# Java API News

## Managment

```
import java.lang.management.*;
import java.util.*;
import javax.management.*;

public class MemTest {
    public static void main(String args[]) {
        List<java.lang.management.MemoryPoolMXBean> pools
            = ManagementFactory.getMemoryPoolMXBeans();
        for(MemoryPoolMXBean p: pools) {
            System.out.println("Memory type="+p.getType());
            System.out.println("Memory usage="+p.getUsage());
        }
    }
}
```

# Sonstiges

- Links:

<http://java.sun.com/developer/technicalArticles/releases/j2se15/>

<http://java.sun.com/j2se/1.5.0/>

<http://www.devchannel.org/devtoolschannel/04/04/29/1549221.shtml>

[http://www-106.ibm.com/developerworks/views/java/articles.jsp?sort\\_order=desc&expand=&sort\\_by=Date&show\\_abstract=true&view\\_by=Search&search\\_by=taming%20tiger%3A](http://www-106.ibm.com/developerworks/views/java/articles.jsp?sort_order=desc&expand=&sort_by=Date&show_abstract=true&view_by=Search&search_by=taming%20tiger%3A)

<http://www.tyma.com/tymajdk15.pdf>

# Ende der 2. Übung

