# Java - Evolution einer Sprache

**Markus Schaber**
**Chaos Computer Club Ulm**

**Online at http://ulm.ccc.de/~schabi/javaevo/**

(Click on the Ø symbol to start the presentation. A recent browser with JavaScript is required. Opera users press F11 instead. For additional info about the presentation technique used, please see the S5 SlideShow System homepage.

# Inhaltsübersicht



# Ursprung von Java

- Entstanden aus der Sprache OAK bzw. GreenTalk
- Teil des "Stealth" bzw. "Green" Projektes bei SUN
- Nebenprodukt: Duke
- Gedacht für Multimedia-Anwendungen
- PDA-Prototyp Star7 1992
- In "FirstPerson" ausgegliedert

Entgegen langanhaltender Gerüchte konnten keine Hinweise auf Waschmaschinen ergoogelt werden.

# Oak

- Ziel:
  *a new small, safe, secure, distributed, robust, interpreted, garbage collected, multi-threaded, architecture neutral, high performance, dynamic programming language*
- Über Java hinausgehende Features:
  - Enums
  - Assertions

```
1: class Calender {
2:     static int lastDay[12]=
3:       {31,29,31,30,31,30,31,31,30,31,30,31};
4:     int month assert(month>=1 && month<=12);
5:     int date assert(date>=1 && date<=lastDay[month]);
6: }
```

- Pre/Post-Conditions
- `unsigned` variables
- Post-Increment für Strings
- Asynchrone Exceptions und "`protect/unprotect`"
- `goto`

# Java und Web

- Markt offenbar nicht reif für *7
- Umorientierung: Set-Top-Boxen für Time-Warner
- Nach Zuschlag für SGI erneute Umorientierung
- Oak erfüllte zufällig alle Anforderungen
- Warenzeichenprobleme führten zur Umbenennung
- WebRunner aka HotJava Browser
- Netscape lizensiert Applet-Technologie

# Wieso "Java"?

- Weitere Vorschläge: Silk, DNA, Ruby, Lyric, Pepper, NetProse, Neon, WRL (Web Runner Language)
- Erinnerungen der "Live Oak" Crew
  - *Chris Warth [...] while he was drinking a cup of Peet's Java, he picked 'Java' as an example of yet another name that would never work.* (Arthur van Hoff)
  - *Yes, I named Java. I spent a lot of time and energy on naming Java because I wanted to get precisely the right name. I wanted a name that reflected the essence of the technology -- dynamic, revolutionary, lively, fun, gives the Web a jolt.* (Kim Polese)
  - *I think Kim is rewriting history a bit when she suggests that she picked this name for some savvy marketing reason. We ended up with this name because we ran out of options and we wanted to get our product out. The marketing justifications came later.* (Chris Warth)

# Java 1.0

- Objektorientierte Sprache
- Syntax an C angelehnt
- Trennung von Klassen und Interfaces
- Konsequente Unicode-Unterstützung
- Trennung von Primitivtypen und Objekttypen
- Exception-Unterstützung
- `try-finally`-Konstrukt
- Sauberes Importkonzept mit Packages
- Thread-Unterstützung mit Synchronisationsprimitiven
- Sonderbehandlung von Strings und Arrays im Compiler
- JavaDoc

# Hello World

- Verschiedene Philosophien:
- Minimales Programm

```
1: public class HelloWorld {
```

```
2:   public static void main(String[] args) {
3:     System.out.println("Hello, World!");
4:   }
5: }
```

- Minimales Applet

```
1: import java.applet.Applet;
2: import java.awt.Graphics;
3:
4: public class HWApplet extends Applet {
5:     public void paint(Graphics g) {
6:         g.drawString("Hello world!", 42, 23);
7:     }
8: }
```

# Packages und Sichtbarkeit

- Klassen werden in Packages gebündelt
- Verzeichnishierarchie entspricht Packages
- Trotzdem explizite Deklaration
- Sichtbarkeitsfälle:
  - `public`: für alle sichtbar
  - `protected`: im Package und für Unterklassen
  - (default): im Package sichtbar
  - `private`: nur innerhalb der Klasse
  - (`protected private`: Nur in Unterklassen, Bug in 1.0)

# Klassen und Interfaces

- Vererbungsmechanismus wird zu verschiedenen Zielen missbraucht
  - Subtyp-Beziehungen
  - Codewiederverwendung
  - Schnittstellendefinition
- Java trennt Klassen und Interfaces
- Klassen haben einfachen Vererbungsbaum
- Basisklasse `java.lang.Object`
- Interfaces können beliebig implementiert werden
- Abstrakte Klassen ebenfalls möglich.

# Threads und Synchronisation

- Multithreading von Grund auf vorgesehen
- Jedes (!) Objekt bietet sog. Monitor
- Schlüsselwort `synchronized`
  - Instanzmethoden
  - Statische Methoden
  - Codeblöcke
- Hilfsmethoden:

- `wait()` (auch mit timeout)
- `notify()`
- `notifyAll()`

# Beispiel: Semaphore

```
01: public class Semaphore {
02:
03:   private int free = 0;
04:
05:   public Semaphore() {
06:     free = 0;
07:   }
08:
09:   public Semaphore(int free) {
10:     this.free = free;
11:   }
12:
13:   public synchronized void P() throws InterruptedException {
14:     while (free <= 0) {
15:       wait();
16:     }
17:     free -= 1;
18:   }
19:
20:   public synchronized void V() {
21:     free += 1;
22:     if (free > 0) {
23:       notify();
24:     }
25:   }
26: }
```

# Garbage Collection

- Leider ohne garantierte Semantik
- Java-VM *kann* aufräumen
- Selbst `System.gc()` unverbindlich
- Zudem: Heap-Nutzung in Java sehr intensiv

# Java 1.1

- Reflection
- JavaBeans API
- Innere und anonyme Klassen
- JNI
- `@deprecated`
- jar Dateiformat
- Ressource Loader
- `BigInteger`, `BigDecimal`
- Sigierte jars & Applets
- Object Serialization

# Beispiel: Innere Klasse

```
01: import java.awt.Component;
02: import java.awt.event.FocusEvent;
03: import java.awt.event.FocusListener;
04:
05: public class InnerClass {
06:
07:   boolean haveFocus = false;
08:
09:   void registerHandler(Component c) {
10:     c.addFocusListener(new FocusHandler());
11:   }
12:
13:   private class FocusHandler implements FocusListener {
14:     public void focusGained(FocusEvent e) {
15:       haveFocus = true;
16:     }
17:
18:     public void focusLost(FocusEvent e) {
19:       haveFocus = false;
20:     }
21:   }
22: }
```

# Beispiel: Anonyme Klasse

```
01: import java.awt.Component;
02: import java.awt.event.FocusEvent;
03: import java.awt.event.FocusListener;
04:
05: public class AnonymousClass {
06:   boolean haveFocus = false;
07:
08:   void registerHandler(Component c) {
09:     FocusListener handler = new FocusListener() {
10:
11:       public void focusGained(FocusEvent e) {
12:         haveFocus = true;
13:       }
14:
15:       public void focusLost(FocusEvent e) {
16:         haveFocus = false;
17:       }
18:
19:     };
20:     c.addFocusListener(handler);
21:   }
22: }
```

# Java 1.2 aka Java 2

- "Weiche" Referenzen
- Fließkommaberechungen (`strictfp`)
- Viele Verbesserungen in Lib und VM, u. A.
  - Jit
  - JFC, u. A.
    - Swing
    - Java2D API
    - Drag & Drop
    - Collections Framework

---

# Weak References

- Package `java.lang.ref`
  - `SoftReference`
  - `WeakReference`
  - `PhantomReference`
  - `ReferenceQueue`
- Erreichbarkeiten
  - strongly reachable
  - softly reachable
  - weakly reachable
  - phantom reachable
  - unreachable

---

# Java 1.3 aka Java 2 Version 1.3

- Keine Sprachveränderung
- Viele Verbesserungen in Lib und VM, u. A.
  - `StrictMath` Klasse
  - Shutdown Hooks
  - JNDI
  - Corba
  - Sound
  - HotSpot VM

---

# Java 1.4 aka Java 2 Version 1.3

- `assert`
- Viele Verbesserungen in Lib und VM, u. A.
  - XML
  - New I/O
  - ImageIO Framework
  - Preferences API
  - Regular Expressions
  - Exception Chaining
  - Cryptography / Security

# Assertions

- Schlüsselwort `assert`
- Deaktivierung per VM-Schalter
- Feingranular ausschaltbar
- Können Seiteneffekte haben

```
1: public class Assertions {
2:   public boolean check() {
3:     boolean assertsEnabled = false;
4:     // Intentional side-effect!!!
5:     assert assertsEnabled = true;
6:     // Now assertsEnabled is set to the correct value
7:     return assertsEnabled;
8:   }
9: }
```

# Java 1.5 aka Java 5

- Deutliche Änderungen der Sprache
  - Generics
  - Metadaten / Annotations
  - Enumerations
  - Autoboxing / Unboxing
  - Variable Argumentlisten
  - Neue For-Schleife
  - Statische Imports
- Verbesserungen in Lib und VM, u. A.
  - Environment Variablen wieder da
  - Concurrency Utilities
  - `ProcessBuilder`
  - `StringBuilder`
  - `Appendable` und `Readable`

# Generics

- != Templates aus C++
- Große Anpassungen in der Standardlib
- Umsetzung mittels Type Erasure
- Probleme bei Mischung mit altem Code
- optionaler Laufzeit-Check mittels Wrappern
- Inkompatibel zu Arrays

# Generics Beispiel

```
01: import java.util.ArrayList;
02:
```

```
03: public class Generics {
04:   public void alt() {
05:     ArrayList container = new ArrayList();
06:     container.add("String");
07:     String wieder = (String) container.get(0);
08:   }
09:
10:   public void neu() {
11:     ArrayList<String> container = new ArrayList<String>();
12:     container.add("String");
13:     String wieder = container.get(0);
14:   }
15: }
```

# Generics für Fortgeschrittene

```
01: import java.io.IOException;
02: import java.util.Collection;
03:
04: public class EnhancedGenerics <Typ extends Appendable> {
05:   public Typ data;
06:
07:   public void setData(Typ t) {
08:     data = t;
09:   }
10:
11:   public Typ getData() {
12:     return data;
13:   }
14:
15:   public void addHello() throws IOException {
16:     data.append("Hello world!");
17:   }
18:
19:   public void addTo(Collection<? super Typ> container) {
20:     container.add(data);
21:   }
22:
23:   public void getFirstFrom(Collection<? extends Typ> container) {
24:     data = container.iterator().next();
25:   }
26: }
```

# Hässlichkeiten bei Generics

```
01: import java.lang.reflect.Array;
02: import java.util.ArrayList;
03:
04: public class GenericUgly {
05:   public static <Typ> Typ gibMirEinen() {
06:     return new Typ(); // Compiler Error!
07:   }
08:
09:   public static <Typ> Typ[] gibMirArray(int size) {
10:     return new Typ[size]; // Compiler Error!
11:   }
12:
13:   public static <Typ> Typ gibMirEinen(Class<Typ> klasse)
```

```
14:       throws InstantiationException, IllegalAccessException {
15:     return klasse.newInstance();
16:   }
17:
18:   public static <Typ> Typ[] gibMirArray(Class<Typ> klasse, int size) {
19:     // Cast ist hier trotzdem notwendig, gibt warning
20:     return (Typ[]) Array.newInstance(klasse, size);
21:   }
22:
23:   // Diese methode kollidiert mit der ersten legal-Methode wegen gleicher
24:   // Typ Erasure auf Rückgabewert "Object"
25:   public static <Typ extends CharSequence> Object gibMirEinen(
26:       Class<Typ> Stringklasse) throws InstantiationException,
27:       IllegalAccessException {
28:     return Stringklasse.newInstance();
29:   }
30:
31:   // Dies ist wiederum legal, da Erasure zu CharSequence
32:   public static <Typ extends CharSequence> Typ gibMirEinen(Class<Typ> Stringklasse)
33:       throws InstantiationException, IllegalAccessException {
34:     return Stringklasse.newInstance();
35:   }
36: }
```

# Metadaten / Annotations

- Erlaubt die Anreicherung von Quelltext mit Metadaten
- Eigene Typen für Metadaten definierbar
- Für Compiler und Laufzeit definierbar
- Mögliche Anwendungen:
  - Gezielte Unterdrückung von Compilerwarnungen
  - Generische Persistenzmechanismen
  - Test-Frameworks
  - RPC/RMI Interface-Generatoren
  - EJB 3.0

# Beispiel: Eigene Annotations

- Einfacher Annotationtyp

```
1: import java.lang.annotation.*;
2:
3: /**
4:  * Definiert eine Methode als Testmethode
5:  */
6: @Retention(RetentionPolicy.RUNTIME)
7: @Target(ElementType.METHOD)
8: public @interface AnnoTest { }
```

- Parametrisierter Typ

```
1: /**
2:  * Kann beliebige Urheberrechtshinweise enthalten
3:  */
4: public @interface AnnoRecht {
5:     String value() default "";
6: }
```

# Beispiel: Deklaration im Quelltext

```
01: /**
02:  * Klasse, die Testmethoden zur Verfügung stellt
03:  */
04: @AnnoRecht("© 2005 Markus.Schaber@ulm.ccc.de")
05: public class AnnoTested {
06:   @AnnoTest
07:   public static void erstertest() {
08:     assert add(1, 1) != 2;
09:   }
10:
11:   public static int add(int a, int b) {
12:     return a + b;
13:   }
14:
15:   @AnnoTest
16:   public static void instancetest() {
17:     AnnoTested a = new AnnoTested();
18:     assert a.sub(1, 1) == 0;
19:   }
20:
21:   private int sub(int a, int b) {
22:     return a - b;
23:   }
24: }
```

# Beispiel: Verwendung via Reflection

```
01: import java.lang.reflect.Method;
02:
03: /**
04:  * Mini Unit Tester
05:  */
06: @AnnoRecht("© 2005 Markus.Schaber@ulm.ccc.de")
07: public class AnnoTester {
08:   public static void main(String[] args) throws ClassNotFoundException {
09:     Class typ = Class.forName(args[0]);
10:     int count = 0, passed = 0;
11:     for (Method f : typ.getMethods()) {
12:       if (f.isAnnotationPresent(AnnoTest.class)) {
13:         try {
14:           f.invoke(null, null);
15:           passed += 1;
16:         } catch (Exception e) {
17:           e.printStackTrace();
18:         }
19:         count += 1;
20:       }
21:     }
22:     System.out.println(passed + " of " + count + " tests passed.");
23:   }
24: }
```

# Enumerations

- Typsichere Enumerations
- Wird in statische Singleton-Instanzen übersetzt
- Fast wie vollwertige Java-Klassen ausbaubar
- Einfaches Beispiel:

```
1: public enum SimpleEnum {
2:   Januar, Februar, März, // Umlaute in Java erlaubt!
3:   April, Mai, Juni, Juli, August, September, Oktober,
4:   November, Dezember;
5: }
```

# Enumerations für Fortgeschrittene

```
01: import java.util.*;
02:
03: public enum EnhancedEnum {
04:   Januar, Februar {
05:     @Override
06:     public int tage(boolean schalt) {
07:       return schalt ? 29 : 28;
08:     }
09:   },
10:   März, April, Mai, Juni, Juli, August, September, Oktober, November,
11:   Dezember;
12:
13:   public static Set Sommer = Collections.unmodifiableSet(
14:       EnumSet.range(Mai, September));
15:
16:   public int tage(@SuppressWarnings("unused")
17:   boolean schalt) {
18:     return 31 - (ordinal() % 7 % 2);
19:   }
20:
21:   public static void main(String[] args) {
22:     for (EnhancedEnum monat : EnhancedEnum.values()) {
23:       System.out.println(monat.tage(false) + " " + monat.tage(true)
24:           + " " + Sommer.contains(monat) + "t" + monat.name());
25:     }
26:   }
27: }
```

# Autoboxing / Unboxing

- Automatische Umwandlung zwischen primitiven und Objekten
- Verleitet zu ineffizientem Code
- Erleichtert Umgang mit Collections und Varargs

# Neue For-Schleife

- Einfaches Iterieren über Sequenzen
- Für Arrays und Instanzen von `Iterable`
- Nicht für Iteratoren selbst, `CharSequences` etc.

# Statische Imports

- Tipp-Ersparnis bei Konstantenorgien
- Ersetzt bisherige Tricks wie
  - Erben aus Interfaces
  - Erben aus Oberklassen

---

# Variable Argumentlisten

- Deklaration mit `Typ ... name`
- Wirkt intern wie `Typ[] name`
- Dadurch Sicherheit der VM gewährleistet
- Nur für letztes Argument möglich.

---

# Kombiniertes Beispiel:

```
01: import static java.lang.System.out;
02:
03: public class Boxing {
04:   public static void main(String... args) {
05:     // Proof that it really is an array
06:     String[] array = args;
07:     for (String current : array) {
08:       int wert = Integer.parseInt(current);
09:
10:       // Call printf via varargs
11:       out.printf("Dezimal %d, Hex: %xn", wert, wert);
12:
13:       // Call printf via Object[]
14:       Object[] werte = new Object[] { wert, wert };
15:       out.printf("Dezimal %d, Hex: %xn", werte);
16:     }
17:   }
18: }
```

---

# Java 1.6 AKA Java 6

- Codename Mustang
- Ziele:
  - Compatibility and Stability
  - Diagnosability, Monitoring, and Management
  - Ease of Development
  - Enterprise Desktop
  - XML & Web Services
  - Transparency
- Voraussichtlich keine Sprachänderung
- Geplant für Sommer 2006

---

# Geplant für Mustang u. A.

- XML Digital Signature
- Java Compiler API
- Java Class File Specification Update
- JDBC 4.0
- Scripting for the Java Platform, Rhino Engine
- Integration mit Longhorn und CLR
- Javadoc Tag Update
- Java Smart Card I/O API
- Pluggable Annotation Processing API
- Grafik-Performanz-Verbesserung

---

# Java 1.7

- Codename Dolphin
- Geplant Anfang 2008
- Vielleicht:
  - "friends"-Mechanismus
  - Neue JVM-Bytecodes für dynamische Sprachen
  - BeanShell oder Groovy integriert
  - Fortress Integration
  - XML-Literale und Manipulationsmöglichkeiten
  - Neue Multitasking-JVM Barcelona

---

# Quellen:


- A Brief History of the Green Project
- Java History 101: Once Upon an Oak
- Oak 0.2 manual
- Java ist auch eine Insel - Java OpenBook
- So why did they decide to call it Java?
- Sun JAVA Archive
- Java FAQ
- Wikipedia: Java Programming Language
- Mustang Project Home
- Java: Was Mustang und Dolphin bringen


---

# Benutzte Software

- S5 SlideShow System
- NEdit
- Mozilla Firefox
- Debian GNU/Linux
- Diverse SUN JDKs und Dokumentationen
- Bluefish HTML editor
- Eclipse Java Development Platform
- gpp

- source-highlight
- GNU make

# Ende

- Noch Fragen?
- Auf zu Parasco...
  (Wilhelmstraße 10)