

Das Java-Konzept: Virtuelle Maschine und Binärdateien

Neue Möglichkeiten für
aktive Online-Dokumentation



Patricia Hallstein, Axel Kramer
Patricia.Hallstein@munich.netsurf.de
axel@well.com



Überblick

- Java in Technischer Dokumentation
- Das Java-Konzept
 - Java's Design
 - Was passiert im Detail?
 - Wie läßt sich das Wissen anwenden?
- ➔ Chancen und Risiken?

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



2



Java! – Java?

<HTML>Dok
</HTML>

Java-Applet

Java-Anwendung

HotJava

JavaBeans

JavaScript

Mocha

Café

Java-VM

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

3

Java in Dokumentation: Wo?

<HTML>Dok
</HTML>

Wo? In HTML-basierten Online-Dokumenten!

HTML?
Im Internet also?

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

4



HTML-Historie

<HTML>Dok
</HTML>

- Erste Generation
 - HTTP
 - einfaches HTML
- Zweite Generation
 - Erweitertes HTML
 - Plug-Ins
 - CGI & Server-Erweiterungen
- Dritte Generation
 - Scripts:
 - JavaScript
 - VBScript
 - Java-Applets
 - ActiveX

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



5

HTML-Historie

<HTML>Dok
</HTML>

- Vierte Generation
 - Style Sheets, Dynamic HTML
 - Distributed Objects
 - »Push«-Techniken
- Wie geht's weiter?
 - Ⓞ Die Basis: Standardisierung
 - Ⓞ Die Perspektive: jenseits des WWW

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



6



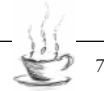
HTML-Standard – Versionen

<HTML>Dok
</HTML>

- Key Players: Netscape – W3C – Microsoft

1989-94 Erfindung des WWW am CERN (Genf)	Nov. 1995 Netscape Navigator 2.0: Java-Applets, JavaScript, Plug-Ins
Juli 1994 W3 Consortium	Nov. 1995 Microsoft Explorer
Okt. 1994 Netscape Navigator 1.0	April 1996 Netscape Navigator 3.0
Nov. 1994 HTML 2.0	May 1996 MS Explorer 3.0
März 1995 Netscape Navigator 1.1	Mai 1996 W3C veröffentlicht HTML 3.2
März-Sept. 1995: HTML 3.0-Vorschlag	1997 MS Internet Explorer 4.0 Netscape Navigator 4.0

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



7

HTML-Perspektive

<HTML>Dok
</HTML>

- Integration in den Desktop

- Netscape Constellation
- Microsoft ActiveDesktop

- ➔ Desktop-Konfiguration durch HTML,
Push-Techniken

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



8



Java in Dokumentation: Wo?

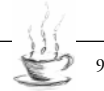
<HTML>Dok
</HTML>

Na dann:

HTML-basierte Online-Dokumente überall!

Und was wird aus
SW-Hilfesystemen?

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



9

Dokumente mit & ohne HTML

<HTML>Dok
</HTML>

lokale Hilfesysteme

- Microsoft Winhelp
- UNIX manual pages
- ...

ferne Dokumente

- Internet: FAQs, News-groups, Mailing-Listen
- WWW:
HTML-Dokumente

- Netscape NetHelp
- MS HTMLHelp

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



10

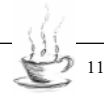


HTML-Dokumente

<HTML>Dok
</HTML>

- Vorteile
 - plattformunabhängig
 - aktuell
 - wirtschaftlich
 - Multimedia
 - Java oder ActiveX
- Nachteile
 - fehlende Navigationshilfen
 - wenig Gestaltungsfreiheit
 - kein kontextabhängiger Aufruf für SW-Dokumentation

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

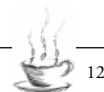


Microsoft HTMLHelp

<HTML>Dok
</HTML>

- **Navigation:** Inhaltsverzeichnis, Index, Volltextsuche, Popup-Fenster, Zweitfenster
- **Fenstergröße und -position festlegbar**
- **Plattformen:** Windows 95/NT, angekündigt: Windows 3.1, Windows for Workgroups, Macintosh, UNIX
- **kontextabhängiger Aufruf**
- **Tools:** einfacher HTML-Editor und Projektverwaltung

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

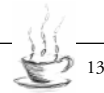


Netscape NetHelp

<HTML>Dok
</HTML>

- Navigation (angekündigt):
 - Index, Volltextsuche, Popup-Fenster, Zweitfenster
- Fenstergröße und -position festlegbar
- kontextabhängiger Aufruf
- Plattformen
 - Windows, UNIX, Macintosh
 - angekündigt: OS/2

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



13

HTMLHelp/NetHelp: Perspektiven

<HTML>Dok
</HTML>

... für Online-Dokumentation:

- Multimedia , Animation, Interaktivität
- Broadcast/Narrowcast News (»push«)
- flexiblere Verteilung und Aktualisierung der Dokumente
- Kommunikation, Personalisieren
- Integration von Dokumentation und SW-Anwendung

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



14

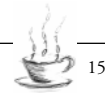


Java-Beispiele und -Ideen

<HTML>Dok
</HTML>

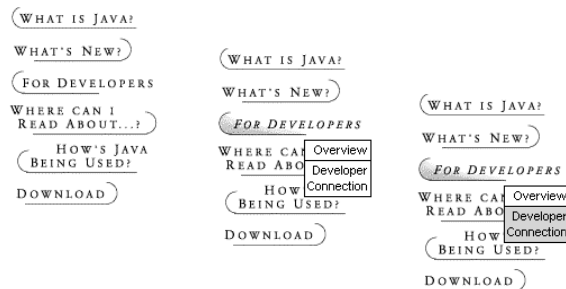
- Navigation
- Animation & Interaktivität
- flexible Verteilung und Aktualisierung
- Kommunikation & Personalisieren
- Integration mit der Anwendung

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



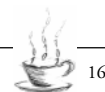
mit Java: Navigation 1

<HTML>Dok
</HTML>



<http://www.javasoft.com>

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



mit Java: Navigation 2

<HTML>Dok
</HTML>

CBS NEWS

Sunday Morning

Up To The Minute

Campaign 96

Eye On The Net @ CBS

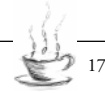
Use the Zoom Bar to navigate.
Double-click on a circle to open that page.

Double Click on nodes to view page in Browser. Drag to pan view.

merzcom I'm Lost! Zoom: [Progress Bar]

<http://www.merzcom.com>

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



17

mit Java: Animation & Interaktivität

<HTML>Dok
</HTML>

Welcome to JambaKid's Prehistoric timeline challenge. I, Professor Spectacle, will use my time tunnel to bring prehistoric animals from the past to the present. Your challenge is to drag each prehistoric animal to the spot on the timeline where it first appeared thus returning it back to it's correct time. You will have 60 seconds to return 15 prehistoric animals. You'll be awarded one gold star for each animal you return. You can click on me for more information about each animal. Good luck!

Play Easy Game

Time 2

PALEOZOIC									
Cambrian	Ordovician	Silurian	Devonian	Carboniferous	Permian				
570-500	500-430	430-335	335-345	345-280	280-245				
Million Years Ago									
MESOZOIC			CENOZOIC						
Triassic	Jurassic	Cretaceous	Paleocene	Eocene	Oligocene	Miocene	Pliocene	Pleistocene	Holocene
245-208	208-145	145-65	64-54	54-38	38-26	26-7	7-2	2m-10,000	Today
Million Years Ago									

<http://www.jamba.com>

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



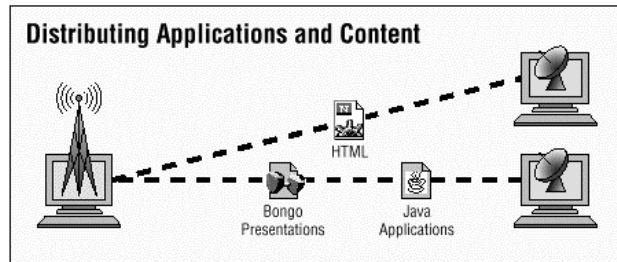
18



mit Java: Verteilung und Aktualisierung

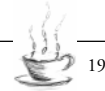
<HTML>Dok
</HTML>

- Beispiel: Castanet (Marimba Inc.)



<http://www.marimba.com>

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



mit Java: Verteilung und Aktualisierung

<HTML>Dok
</HTML>

Ein Szenario:

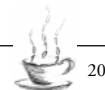
Mit ihrer neuen Software erhalten sie eine Online-Dokumentation mit den **Kerninhalten auf CD**.

Updates und Erweiterungen dazu werden über Internet abgerufen und auf Ihrer Festplatte gespeichert – automatisch nachts oder sobald Sie eine Internet-Verbindung aufgebaut haben.

Die Online-Dokumentation zeigt immer die aktuellste verfügbare Version an und **weist Sie auf wichtige neue oder geänderte Inhalte hin**.

Durch **Verweise ins Internet** erhalten Sie die Informationen zu besonderen Themen, die (noch) nicht im lokalen System vorhanden sind.

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

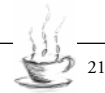


mit Java: Kommunikation & Personalisieren

<HTML>Dok
</HTML>

- Chat: moderierte Diskussion oder Unterricht
- Antwort-Roboter
 - mit persönlichem Profil
 - mit Verknüpfung mit Helpdesk
 - Server- oder Client-basiert

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



mit Java: Integration mit der Anwendung

<HTML>Dok
</HTML>

Ein Szenario:

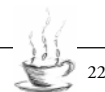
Hilfesystem und Software-Anwendung sind enger miteinander verbunden als heute möglich:

Eingaben für die laufende Anwendung kann die Benutzerin im Hilfesystem machen.

Kontextbezogene Hilfen werden direkt neben, über oder unter der Aufrufstelle angezeigt.

Voraussetzungen: Das Hilfesystem kann die Anwendung steuern und eine Programmierschnittstelle ist für die Anwendung offengelegt.

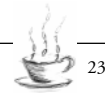
t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



Überblick: Java

- Java's Design
 - Die Herausforderung – Alte Technologien – Design-Alternativen – Die Entscheidung
- Was passiert im Detail?
 - Binärcode – Virtuelle Maschine – Ausführung
- Wie läßt sich das Wissen anwenden?

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



23

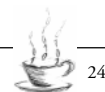
Ausblick

- Übertragungsformat verstehen
- Ausführungsmechanismus verstehen

- Risiko beurteilen können
- Flexibilität nutzen können



t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



24

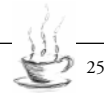


Javas Design

- Die Herausforderung
- Alte Technologien
- Design-Alternativen
- ➔ Die Entscheidung

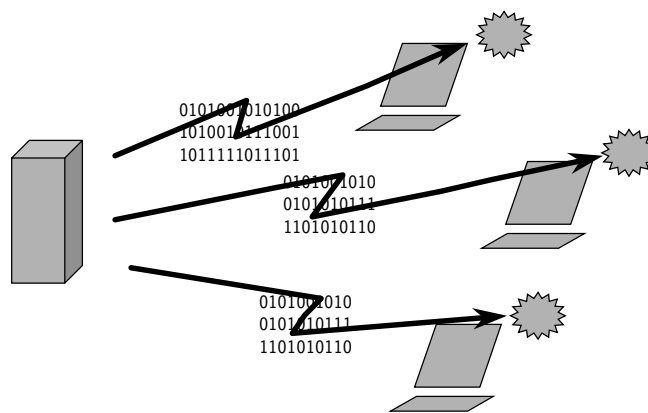


t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



25

Die Herausforderung



t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

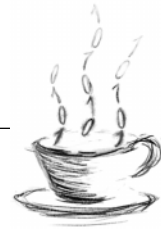


26

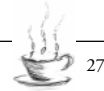


Alte Technologien

- spezifisch für eine Plattform
 - große, gebundene Komponenten
 - Versionsprobleme bei der Vererbung
- ⇒ ungeeignet für Netzwerk-Applikationen
viele Plattformen – langsamer Transport – schnelle Zyklen



t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



27

Alternativen

Basis: objektorientierte Sprache! OOP

- Welche Zielmaschine?
- Welche Art von Typen?
- Welche Ausführungseinheiten?
- Welche Speicherverwaltung?



t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

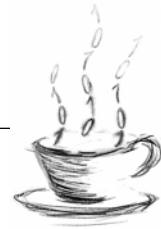


28

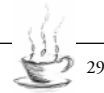


Welche Zielmaschine?

- Plattformabhängigkeit
 - Programmgröße
 - Effizienz
- ⇒ Virtuelle Maschine (Bytecode)
oder konkrete CPUs (Maschinencode)?



t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



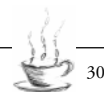
29

Welche Art von Typen?

- Klarheit
 - Sicherheit
 - Programmgrenzen
 - Effizienz
- ⇒ statische oder dynamische Typen?



t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

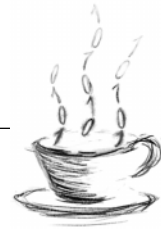


30

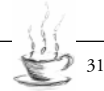


Welche Ausführungseinheiten?

- Größe
 - Wiederverwendbarkeit
 - Programmversionen
- ⇒ modulare oder monolithische Struktur



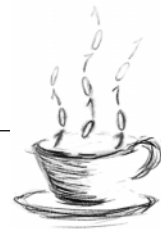
t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



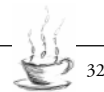
31

Welche Speicherverwaltung?

- Einfachheit
 - Ökonomie
 - Speicherfreigabe
 - Effizienz
- ⇒ automatische oder manuelle Speicherverwaltung



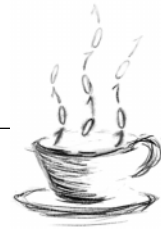
t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



32



Die Entscheidung






- Bytecodes
 - statische Typen
 - modulare Struktur
 - automatische Speicherverwaltung
- ⇒ ausführliche Binärdateien
- ⇒ »Tricks« ⇒ sehr spätes Binden
- ⇒ Bytecodes mit Typinformation

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

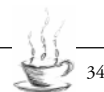


33

Was passiert im Detail?

- Binärdateien 
Woraus bestehen sie und warum sind sie so aufgebaut?
- Virtuelle Maschine (VM) 
Wozu gibt es sie und wie funktioniert sie?
Wie ist der Befehlssatz aufgebaut (Bytecodes)?
- ⇒ Ausführung 
Wie paßt das alles zur Laufzeit zusammen?

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



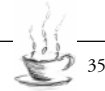
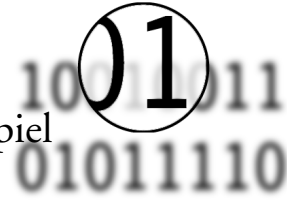
34



Binärdateien

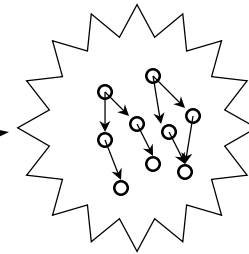
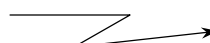
- Anforderungen
- Überblick und vereinfachtes Beispiel
- Ausgewählte Details

Konstanten – Deskriptoren – Variablen – Methoden –
Typinformationen

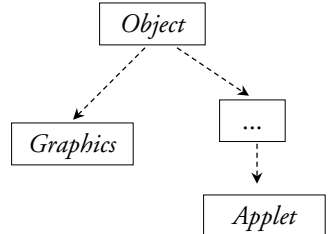


Anforderungen

```
...  
0101001010100  
1010010111001  
1011111011101  
...
```



```
public class HelloWorld  
    extends java.applet.Applet {  
    ...  
    public void paint(Graphics g) {  
        g.drawString("Hello world!", 50, 25);  
    }  
    ...  
}
```

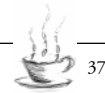


Überblick

01
1001011
01011110

- eine Binärdatei pro Klasse mit:
 - Konstanten
 - Zugriffsrecht (»access flag«)
 - Name der Klasse
 - Name der Superklasse
 - Interfaces
 - Variablen
 - Methoden
 - Attributen

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



37

Vereinfachtes Beispiel

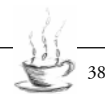
01
1001011
01011110

```
...  
0101001010100  
1010010111001  
1011111011101  
...
```

```
public class HelloWorld  
    extends java.applet.Applet {  
    ...  
    public void paint(Graphics g) {  
        g.drawString("Hello world!", 50, 25);  
    }  
    ...  
}
```

```
...  
Zugriffsrecht    public  
Klasse           HelloWorld  
Superklasse      java.applet.Applet  
...  
Methoden  
Zugriffsrecht    public  
Methode          paint  
Descriptor       (Ljava/awt/graphics;)V  
Attribute  
Code             43 18 1  
                 16 50 16 25  
                 182 0 7 177  
...
```

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



38

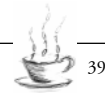


Details



- Konstanten (»constant pool«)
- Deskriptoren
- Variablen (»fields«)
- Methoden
- Typinformationen

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



39

Details: Konstanten (constant pool)



```
import java.awt.Graphics;  
public class HelloWorld extends java.applet.Applet  
{  
    public void init(){resize(150,25);}  
  
    public void paint(Graphics g) {  
        g.drawString("Hello world!", 50, 25);}  
}
```

```
11 name: 27 descriptor: 32  
12 asci: (Ljava/awt/Graphics;)V  
13 asci: resize  
14 asci: (II)V  
15 asci: ConstantValue  
16 asci: Exceptions  
17 asci: LineNumberTable  
18 asci: SourceFile  
19 asci: LocalVariables  
20 asci: Code  
21 asci: drawString  
22 asci: init  
23 asci: java/awt/Graphics  
24 asci: java/awt/Component  
25 asci: java/applet/Applet  
26 asci: HelloWorld.java  
27 asci: <init>  
28 asci: Hello world!  
29 asci: (Ljava/lang/String;II)V  
30 asci: paint  
31 asci: HelloWorld  
32 asci: ()V
```

Konstanten

```
1 string: 28  
2 class: 23  
3 class: 25  
4 class: 31  
5 class: 24  
6 class: 3 method: 11  
7 class: 2 method: 10  
8 class: 5 method: 9  
9 name: 13 descriptor: 14  
10 name: 21 descriptor: 29
```

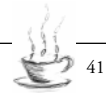
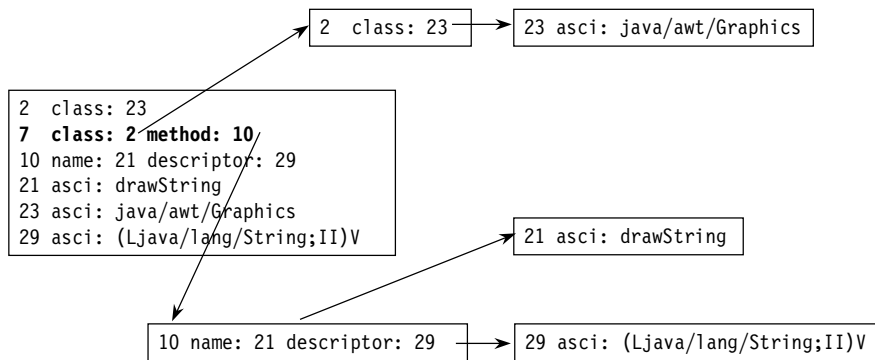
t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



40






Details: Konstanten auflösen

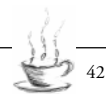


Details: Deskriptoren



Ein Deskriptor beschreibt den Typ:

- einer Variablen 
value [C
offset I
static InternSet Ljava/util/Hashtable;
- eines Arguments 
HelloWorld.paint(Ljava/lang/String;II)V
String.<init> ([CII)V
- eines Rückgabewerts 
String.concat (LString;)LString;
String.regionMatches (ZILString;II)Z



Details: Variablen (»fields«)

01
1001011
01011110

Für jede Klassen- oder Exemplar-Variable einer Klasse wird beschrieben:

- Attribute
- Name
- Deskriptor
- konstanter Wert (wenn notwendig)

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



Details: Methoden

01
1001011
01011110

Für jede Methode der Klasse wird beschrieben:

- Attribute
- Name
- Deskriptor
- Generische Attribute
 - Programmcode
 - Ausnahmebehandlung (exceptions)

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



Details: Typinformationen

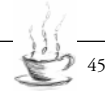


```
public class HelloWorld extends java.applet.Applet {  
    ...  
    public void paint(Graphics g) {  
        g.drawString("Hello world!", 50, 25);  
    }  
    ...  
}
```

Bytecodes

```
paint descriptor: (Ljava/awt/Graphics;)V  
1 <43 >    load object at: 1  
2 <18 1 >   push pool constant: 'Hello world!'  
4 <16 50 >  push byte: 50  
6 <16 25 >  push byte: 25  
8 <182 0 7 > invoke virtual: java/awt/Graphics  
           drawString (Ljava/lang/String;II)V  
11 <177 >   return
```

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



45

Virtuelle Maschine (VM)

- Ausführung
 - Sicheres Laden
 - Laufzeitbereiche
- Datentypen
- Anweisungen – Bytecode
- ◎ Optimierungsmöglichkeiten



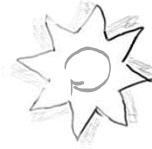
t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



46



VM: Ausführung

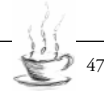


Binärdateien:

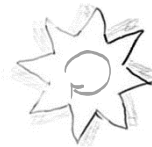
- laden
- überprüfen
- binden
- initialisieren

Bytecodes:

- dekodieren
- Konstanten auflösen
- Typ prüfen
- ausführen



VM: Sicheres Laden



Die VM verifiziert, daß eine Binärdatei mit der Java-Sprachdefinition übereinstimmt: vor dem Einbinden von Binärdateien und vor dem Ausführen von Bytecodes.

- Format der Binärdateien
- Konstanteneinträge
- Eigenschaften von Klassen
- Eigenschaften von Methoden



VM: Laufzeitbereiche

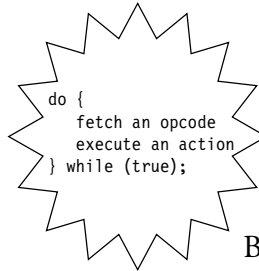


Per Thread

Stack-Einträge – »stack frames«

Programm-Zähler
lokale Variablen
Operanden-Stack

Programm-Zähler
lokale Variablen
Operanden-Stack



Heap



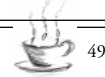
Methoden

```
HelloWorld.draw
43 18 116 50 16
25 182 0 7 177
```

Bindeinformation

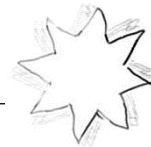
```
HelloWorld =>
...
27 asci: <init>
28 asci: Hello world!
29 asci: (Ljava/lang/String;II)V
...
```

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



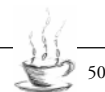
49

VM: Datentypen



- Einfache Datentypen
 - Signed Byte (8), Signed Short (16)
 - Unicode Char (16)
 - Integer (32), Long (64)
 - Float (32), Double (64)
- Object (32)
- Return Address (32)

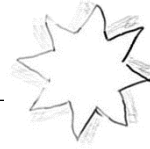
t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



50



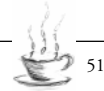
VM: Anweisungen – Bytecodes



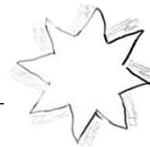
Die Bytecodes der virtuellen Maschine ähneln den Anweisungen einer CPU.

- pro Anweisung 1 Byte für Operanden 0-*n* Bytes
- ➔ arbeiten normalerweise auf dem Stack
- ➔ sind spezifisch für bestimmte Typen

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

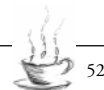


VM: Bytecode-Gruppen



- Arithmetik
- logische Operationen
- Stack-Bearbeitung
- Steuerung
- Konstanten
- Klassen-, Exemplar-, lokale Variablen
- Felder (arrays)
- Aufruf von Methoden
- Rücksprung aus Methode
- Konvertierung
- Diverse

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

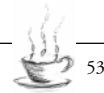


Optimierungsmöglichkeiten



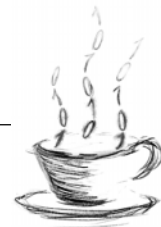
- Dynamisches Ersetzen von Bytecodes
 - Referenzen in Konstantentabelle auflösen, überprüfen,
 - und dann Bytecode durch »quick bytecode« ersetzen
- Just in Time Compiler (JIT)
 - dynamisches Übersetzen von Bytecodes in »native code«

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



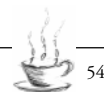
53

Java-VM: Zusammenfassung



- Binärdateien
 - eine Binärdatei pro Klasse
 - symbolische Referenzen
 - Typinformationen
- Virtuelle Maschine
 - spätes Binden
 - dynamisches und sicheres Laden von Binärdateien

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



54

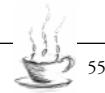


Wissen anwenden: Beispiele

- Bytecodes → Quelltext
- benötigte Klassen finden



t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



55

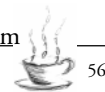
Bytecodes → Quelltext

```
public void strokeEdge(GraphicsContext)
1 <3 >      push integer: 0
2 <61 >     store integer at: 2
3 <3 >      push integer: 0
4 <62 >     store integer at: 3
5 <3 >      push integer: 0
6 <54 4 >   store integer at: 4
8 <3 >      push integer: 0
9 <62 >     store integer at: 3
10 <167 0 98 > goto: 108
13 <42 >    load object at: 0
14 <180 0 16 > get field: FARDrawOutline.flags '[B'
17 <29 >    load integer at: 3
18 <51 >    array load byte
19 <16 6 >   push byte: 6
21 <160 0 31 > integers not equal jump to:52
24 <28 >    load integer at: 2
25 <54 4 >   store integer at: 4
27 <43 >    load object at: 1
28 <42 >    load object at: 0
29 <180 0 17 > get field: FARDrawOutline.points '[I'
32 <28 >    load integer at: 2
33 <132 2 1 > increment local: 2 by: 1
36 <46 >    array load integer
37 ...
```

```
public void strokeEdge(GraphicsContext gc){
    int i = 0; int j = 0; int k = 0;
    for (j = 0; j < flags.length; j++) {
        if (flags[j] == 6) {
            k = i;
            gc.moveTo(points[i++], points[i++]);
        } else {
            gc.edgeTo(points[i++], points[i++]);
            if (flags[j] == 3)
                gc.edgeTo(points[k++], points[k]);
        }
    }
}
```

Mocha: <http://web.inter.nl.net/users/H.P.van.Vliet/mocha.htm>

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



56

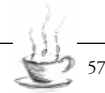


Benötigte Klassen finden



- Binärdatei einer Klasse laden und dann sammeln:
 - Superklasse
 - konstante Klasseneinträge
 - Klassen in Deskriptoren
- für alle noch nicht bearbeiteten gesammelten Klassen wiederholen

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



57

Chancen und Risiken



t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



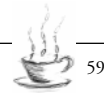
58



Anhang

- Objektorientierte Konzepte
- VM: Bytecode-Formate
- VM: Konstruktoren
- VM: String & StringBuffer
- VM: Aufruf von Methoden

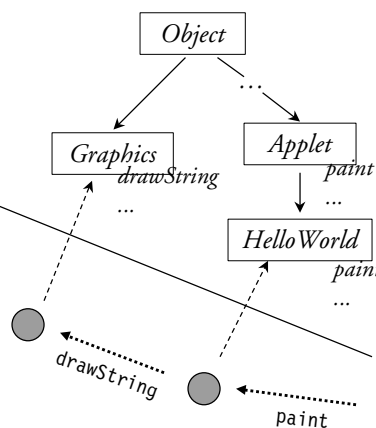
t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



59

Objektorientierte Konzepte

- Klassen
- Methoden
- Objekte
- Nachrichten



t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



60

back

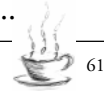


VM: Bytecode-Formate



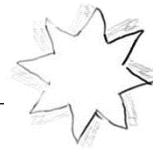
<i>kein Operand</i>	nop = 0	... => ...
<i>konstanter Operand</i>	const_m1 = 2	... => ..., -1
<i>im Opcode</i>	iconst_<n> = <0>3..8	... => ..., n
<i>in extra Byte</i>	bipush = 16, byte	... => ..., byte
<i>Stack-Operand</i>	dup = 89	a => a, a
<i>indirekter Operand</i>	ldc1 = 18, index	... => ..., pool[index]
<i>lokaler Operand</i>	aload_<n> = <0>42..45	... => ..., vars[n]
<i>Mix</i>	iinc = 132, vindex, const	... => ...
<i>offen</i>	tableswitch = 170	..., index => ...

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



61

VM: Konstruktoren

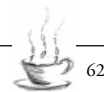


```
import java.awt.Graphics;
public class HelloWorld extends java.applet.Applet {
    public void init(){resize(150,25);}

    public void paint(Graphics g) {
        g.drawString("Hello world!", 50, 25);}
}
```

```
<init> descriptor: ()V
1 <42 >      load object at: 0
2 <183 0 6 >  invoke non virtual: java/applet/Applet.<init>
               type: ()V
5 <177 >      return
```

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



62



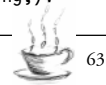
VM: String & StringBuffer



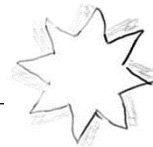
```
int count = 0;
...
System.out.println("Input has " + count + " chars.");
```

```
19 <178 0 14 > get static field: System.out type: Ljava/io/PrintStream;
22 <187 0 9 > new object of: StringBuffer
25 <89 > dup
26 <183 0 17 > invoke non virtual: StringBuffer.<init> type: ()V
29 <18 1 > push pool constant: 'Input has '
31 <182 0 16 > invoke virtual: StringBuffer.append type: (Ljava/lang/String;)Ljava/lang/StringBuffer;
34 <27 > load integer at: 1
35 <182 0 12 > invoke virtual: StringBuffer.append type: (I)Ljava/lang/StringBuffer;
38 <18 2 > push pool constant: ' chars.'
40 <182 0 16 > invoke virtual: StringBuffer.append type: (Ljava/lang/String;)Ljava/lang/StringBuffer;
43 <182 0 13 > invoke virtual: StringBuffer.toString type: ()Ljava/lang/String;
46 <182 0 15 > invoke virtual: java/io/PrintStream.println type: (Ljava/lang/String;)V
```

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer



VM: Aufruf von Methoden



- `invoke virtual` Exemplar-Methoden
- `invoke static` Klassen-Methoden
- `invoke special` Initialisierung, private Methoden, und Methoden der Superklasse
- `invoke interface` Interface-Methoden

t.i.e.m. 97
© 1997 Patricia Hallstein, Axel Kramer

