

Themenübersicht

1. JAVA, was ist das
2. Installation
3. Compilieren
4. Applikation starten
5. Applikation debuggen
6. JAR - Pakete
7. Plattformunabhängigkeit ?
8. Literatur
9. Wünsche
10. Fragen

1. JAVA, was ist das

- keine Compiler-, sonder Interpretersprache
Compiler: Vorbereitung der Interpretation
Linker-Checks: nicht auflösbare Referenzen
- objektorient; sehr viel Ähnlichkeit mit C++
- äußerst mächtige Library (API)
- z.B. leistungsfähige Desktop-Oberflächen mit wenig Aufwand
- Kennenlernen der API: wichtig und zäh

1. JAVA

- CLASS

- kapselt Daten und Methoden
- Methode = gekapselte Funktion
- Class ist Generierungsvorschrift, die zur Laufzeit interpretiert wird
- Instanziierung: Anlegen eines Objektes zu einer Klasse
Objekt = instanziierte Klasse
dabei Anlegen und Initialisieren von Speicher

2. Installation JAVA JDK 1.5

- Voraussetzungen
 - ODS-5 formatierte Platte
 - OpenVMS ab 7.3-2
 - aktuelle Patches einspielen
- www.itrc.hp.com – > maintenance and support (Compaq Products)
- > individual patches for OpenVMs and Tru64 UNIX
- * zuerst neueste PSCI-Version (Bug bei der KIT-Validation)
 - * OpenVMS-Update-Patches
 - * SYS-Patches
 - * RMS-Patches
 - * neueste verfügbare TCPIP-Version und ggf. letzte ECOs
 - * evtl. weitere (siehe Patch-List im itrc)

2. Installation JAVA JDK 1.5

- JAVA download

<http://h18012.www1.hp.com/java/download/>

<http://h18012.www1.hp.com/java/download/ovms/1.5.0/index.html>

http://h18012.www1.hp.com/java/download/ivms/1.5.0/jdk5.0_down.html

z.B. auf SYS\$COMMON: [000000]

aktuelle Patch-Hinweise

http://h18012.www1.hp.com/java/download/ovms/1.5.0/jdk5.0_patches.html

- auspacken \$ r DEC-AXPVMS-JAVA150-V0105-3-1.PCSI_SFX_AXPEXE

DEC-AXPVMS-JAVA150-V0105-3-1.PCSI\$COMPRESSED (316185 blocks)

DEC-AXPVMS-JAVA150-V0105-3-1.PCSI\$COMPRESSED_ESW (18 blocks)

Hinweis: ggf. muss PCSI mit dem neuesten Patch aktualisiert werden

2. Installation JAVA JDK 1.5

- Release Notes vorher extrahieren

```
$ prod extract release java150
```

nach der installation: [. JAVA\$150.DOC] RELEASE_NOTES.HTML

- Installation mit

```
$ prod ins JAVA150
```

Die Installation erfolgt standardmäßig in [. JAVA\$150...]

z.B. SYS\$COMMON: [JAVA\$150...]

2. Installation JAVA JDK 1.5

- System- und Prozessparameter anpassen

siehe [`.JAVA$150.DOCS`] `USER_GUIDE.HTML`]

oder Vortrag v. Guy Peleg TUD 2007 Bad Homburg

Environment check mit

```
$ @ [ .JAVA150.COM] JAVA$CHECK_ENVIRONMENT.COM
```

2. Installation JAVA JDK 1.5

- Setup

```
$ @SYS$STARTUP: JAVA$150_SETUP
```

ruft nur z.B. auf:

```
$ @SYS$COMMON: [JAVA$150.COM] JAVA$150_SETUP.COM
```

3 Parameter (siehe JAVA\$150_SETUP.COM)

- ENGINE: CLASSIC (DEFAULT), FAST, HOTSPOT
- LOGICAL-TYPE: PROCESS(DEFAULT) PROCESS_CONFINE, JOB
- SYMBOL_DEF: SYMBOLS(DEFAULT), NOSYMBOLS

2. Installation JAVA JDK 1.5

- zusätzliche JAVA-Parameter setzen (nach SETUP)
(s. USER_GUIDE.HTML)

```
$ @ [. JAVA150 . COM] JAVA$CONFIG_SETUP . COM
```

mit

```
$ @ [. JAVA150 . COM] JAVA$CONFIG_WIZARD . COM
```

wird ein neuer JAVA\$CONFIG_SETUP.COM erzeugt

- Start-Prozeduren in SYS\$MANAGER:SYSTARTUP_VMS.COM
oder SYS\$MANAGER:SYLOGIN.COM einbauen, z.B.:

```
$ @SYS$COMMON: [JAVA$150 . COM] JAVA$150_SETUP . COM
```

und ggf. (in dieser Reihenfolge)

```
$ @SYS$COMMON: [JAVA150 . COM] JAVA$CONFIG_SETUP . COM
```

3. Compilieren

kein Compiler im Sinne von C, Fortran, Cobol etc.
eher Pre-Compiler mit etwas Linker-Eigenschaften

- Syntax (Standard)

```
$ javac beispiel.java
```

- Eingabe ist nicht case-sensitiv

- vollständiger Dateiname mit Extension erforderlich

```
$ set def <directory der java-file>
```

```
$ javac phrasomat.java
```

```
$
```

3. Compilieren

- Directory-Angabe der Quelle wird nicht akzeptiert

```
$ javac DKB0:[JAVA]phrasomat.java
/dkb/java/phrasomat.java:1: class PhrasOMat is public, should
be declared in a file named PhrasOMat.java
public class PhrasOMat
    ^
1 error
$
```

3. Compilieren

- auch der switch `-sourcepath <path>` scheint nicht zu funktionieren:

```
$ javac -sourcepath [JAVA] phrasomat.java    oder
```

```
$ javac -sourcepath /JAVA phrasomat.java
```

```
error: cannot read: phrasomat.java
```

```
1 error
```

```
$
```

3. Compilieren

- Angabe Ziel-Directory für .CLASS-Files

```
$ javac -d [java] phrasomat.java
```

```
$ dir ph*.class;
```

```
Directory DKB0: [JAVA]
```

```
PHRASOMAT.CLASS;4          3/3          26-MAR-2008 16:59:07.74
```

aber: enthält die Quelle die Anweisung `package phrase;`

dann wird ein subdirectory mit dem package-Namen erzeugt

```
$ dir [JAVA...]ph*.class;
```

```
Directory DKB0: [JAVA.PHRASE]
```

```
PHRASOMAT.CLASS;4          3/3          26-MAR-2008 16:59:07.74
```

3. Compilieren

- mehrere Quelldateien

```
$ javac *.java
```

- mit Ziel-Directory-Angabe f. die .CLASS-Files

```
$ javac -d [FHL.MF] *.java
```

- auch hier gilt: bei Vorhandensein eine package-Anweisung werden die .CLASS-Files in dem entsprechenden Unterdirectory erzeugt.

- Unresolved references:

Check immer nur über .java-files mit dem gleichen Package-Namen oder ohne Package-Namen

4. Start einer Application

- Start eine einfachen Applikation - eine Quelldatei
angeben die Start-Klasse, aber case-sensitiv

```
$ java "Phrasomat"
```

Was wir brauchen, ist eine konkurrenzfaehige haftende Schicht
oder

```
$ define decc$argv_parse_style enable
```

```
$ define decc$efs_case_perserve enable
```

```
$ set proc/parse=extended
```

```
$ java PhrasOMat
```

Was wir brauchen, ist eine clevere Mehrwert-Kernkompetenz

```
$
```

4. Start einer Application

- Start einer Applikation mit mehreren Quelldateien (kein package)
(für Dir-Namen gibt es keine Namenskonventionen)

```
$ set def [java.im.cls] = directory der .CLASS-Files
```

```
$ java "IMPR"    die class IMPR enthält die methode main
```

oder aus einem beliebigen directory

```
$ java -classpath [java.im.cls] "IMPR"
```

mit -classpath wird das directory der .CLASS-Files angegeben

4. Start einer Application

- Start einer Applikation mit mehreren Quelldateien (package)

– Voraussetzungen

* jede Quelle enthält die Anweisung `package <name>;`

z.B. `package imPr;`

* das Unter-directory mit den .CLASS-Files MUSS den package-Namen haben; Gross-/Kleinschreibung ist dabei nicht relevant

Beispiel: `package = imPr` `dir = [PROJ.CLASSALL.IMPR]`

4. Start einer Application

- Start einer Applikation mit mehreren Quelldateien (package)

- Start

Anwahl des Directories, das den package-Namen trägt und in dem sich die .CLASS-Files befinden

Directory der .CLASS-Files: [PROJ.CLASSALL.IMPR]

```
$ set def [PROJ.CLASSALL]
```

```
$ java "imPr.IMPR"
```

mit imPr = package-Name und IMPR = class mit Methode main

- Start aus einem beliebigen Directory

```
$ java -classpath [PROJ.CLASSALL] "imPr.IMPR"
```

4. Debuggen

- command-orientiert
- umständlich (“Steinzeit-Debugger“)
- aber immer noch viel besser als print-Ausgabe
- Einschränkungen
 - Debuggen der main-class einfach und unproblematisch
 - Debuggen in andere Threads habe ich noch nicht hinbekommen

4. Debuggen

- Start

```
$ jdb "PhrasOMat"
```

oder

```
$ define decc$argv_parse_style enable
```

```
$ define decc$efs_case_perserve enable
```

```
$ set proc/parse=extended
```

```
$ jdb PhrasOMat
```

```
Initializing jdb ...
```

```
>
```

4. Debuggen

- Start mit falscher Klassenangabe

```
$ jdb "Phrasomat"  Initializing jdb ...
```

```
> run                ! Eingabe run
```

```
run phrasomat
```

```
>
```

```
VM Started: Exception in thread "main" java.lang.NoClassDef
```

```
FoundError: Bad class name (expect: phrasomat, get: PhrasOMat)
```

```
Fatal error
```

```
Failed reading output of child interpreter
```

```
$
```

Fehlermeldung erst nach Start der VM

4. Debuggen

- Eingabe eines Breakpoints

```
> stop in phrasOMat.main
```

```
deferring breakpoint phrasOMat.main
```

```
It will be set after the class is loaded
```

keine Überprüfung, ob es die Klasse überhaupt gibt !!

```
> run
```

```
run PhrasOMat
```

```
>
```

```
VM Started:
```

```
Was wir brauchen, ist eine konkurrenzfaehige haftende Schicht
```

Nach Start läuft das Programm bis zum Ende durch

4. Debuggen

- Prompt des jdb holen

Eingabe eines <Return> bewirkt den jdb-Prompt

```
>
```

```
> exit
```

Mit exit wird das Programm und die jdb-Umgebung verlassen

4. Debuggen

- Eingabe eines korrekten Breakpoints

```
> stop in PhrasOMat.main
```

```
deferring breakpoint phrasOMat.main
```

```
It will be set after the class is loaded
```

```
> run
```

```
run PhrasOMat
```

```
>
```

```
VM Started: Set deferred breakpoint PhrasOMat.main
```

```
Breakpoint hit: "thread=main", PhrasOMat.main(), line=6 bci=0
```

```
6      String[] wortListeEins = {"verlaessliche", .....
```

```
>
```

```
main[1]
```

4. Debuggen

- step bewirkt die Abarbeitung der aktuellen Zeile

```
main[1] step
```

```
>
```

```
Step completed: "thread=main", PhrasOMat.main(), line=7 bci=66
```

```
7      String[] wortListeZwei = {"gepowerte ", "haftende "....
```

```
main[1]
```

- cont bewirkt Programmabarbeitung ab Breakpoint

```
>
```

```
main[1] cont
```

```
>
```

```
Was wir brauchen, ist eine kundenorientierte Mehrwert-Endstufe
```

4. Debuggen: Auswahl wesentlicher Befehle

JDK-BEFEHL	VMS-DEBUG
stop in <class>.<method>	se br <module>
run	go (from the beginning)
next	s
step	s/into
step up	s/ret
cont	go
print <variable>	examine
list <n>	Ausgabe von 10 Zeilen um Zeile n herum
set <variable> = <wert>	deposit

6. JAR-File

- jar = Krug, Tiegel ...
- eine Library für .CLASS-files
- utility jar zum Erzeugen dieser Library
- Direkt-Start einer Application aus einem jar-File
- für Direkt-Start manifest-file erforderlich

zur Angabe der CLASS mit der Methode main

Main-Class: <package=Name>.<Main-class>

Beispiel eines Manifest-Files (eine Zeile, case-sensitive !!)

Main-Class: impr.IMPR

6. JAR-File

- jar-Syntax entspricht nicht ganz der online-Hilfe
- Syntax (Standardbeispiel)

```
$ jar -cvfm <jar-File> <manifest> <package> oder
```

```
$ jar -cvmf manifest.txt impr.jar impr
```

OpenVMS-jar-utility akzeptiert keine UNIX-Style-Directories

Directory-Angabe im DCL-Format

```
$ jar -cvfm [root.xx]impr.jar [dir.subdir]manifest.txt [de.impr]
```

- package-Name == Directory-Struktur

Beispiel: de.impr -> [.de.impr]

6. JAR-File

- jar-File können mit unzip bearbeitet werden

```
$ unzip -l impr.jar
```

```
Archive:  SYS$COMMON: [TOOLS.IMPR] IMPR.JAR;7
```

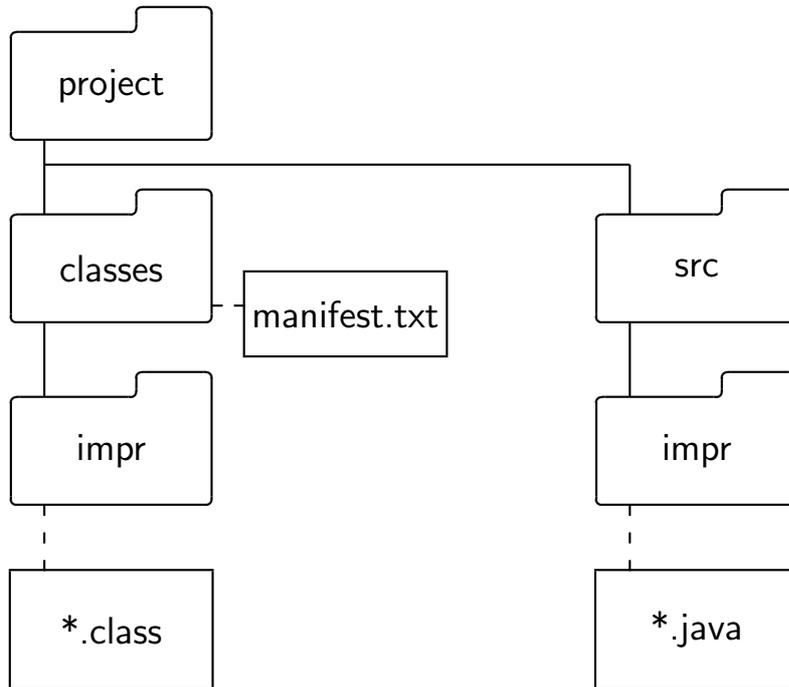
Length	Date	Time	Name
0	03-30-08	22:37	META-INF/
93	03-30-08	22:37	META-INF/MANIFEST.MF
0	03-30-08	22:28	impr/
583	03-30-08	22:28	impr/IMPR\$1.class
4336	03-30-08	22:28	impr/TextEinlesen.class
38442			22 files

```
$
```

- .class-files im jar-file sind per Default compressed

6. JAR-File

- Zweckmäßige Directory-Struktur zum Bilden von jar-Files



6. JAR-File

- Ordner anlegen

```
$ create/dir [java.project.src.impr]
```

```
$ create/dir [java.project.classes]
```

- *.java nach [.project.classes.impr] kopieren

Package-Anweisung in jeder Quelle: `package impr;`

- .class-files erzeugen

```
$ set def [java.project.src]
```

```
$ javac -d [java.project.classes] impr
```

- Code ausführen

```
$ set def [java.project.classes]
```

```
$ java impr
```

6. JAR-File

- Manifest-Datei erzeugen

```
$ [java.project.classes]manifest.txt
```

```
Inhalt: "MAIN-Class: impr.IMPR"
```

- Ausführbares JAR-File erzeugen \$ set def [java.project.classes]

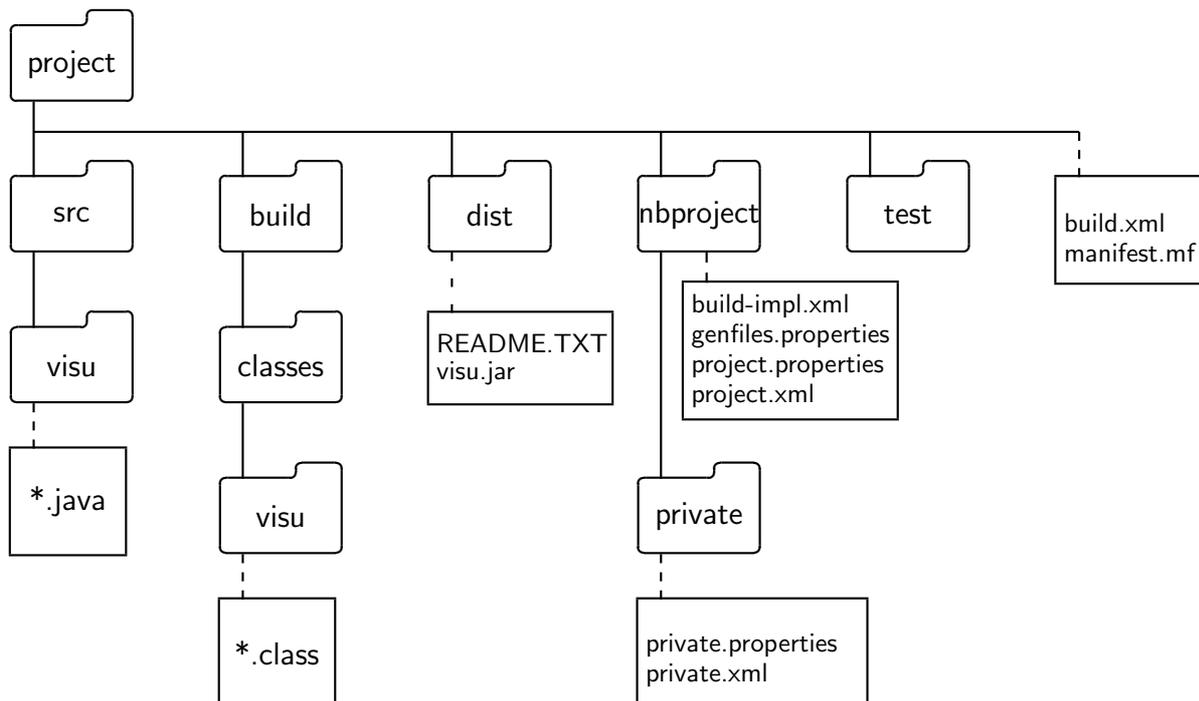
```
$ jar -cvmf manifest.txt imprpak.jar impr
```

- Ausführbares JAR-File starten \$ set def [java.project.classes]

```
$ java -jar imprpak.jar
```

6. JAR-File

- NETBEANS-Dateistruktur eines Projektes



7. Plattformunabhängigkeit

- JAin; 99,9% JA, 0,1% nein
- 1. Beispiel: LINUX <=> WINDOWS/OpenVMS

```
server = new ServerSocket();  
// server.bind(new InetSocketAddress(  
//           InetAddress.getLocalHost(), plcPort));  
server.bind(new InetSocketAddress(plcPort));
```

Die auskommentierte Anweisung erzeugt unter LINUX einen ip6-Eintrag.

Fehlermeldung: Connect to network object rejected

7. Plattformunabhängigkeit

- 2. Beispiel: WINDOWS <=> LINUX/OpenVMS

frame: aus JFrame vererbtes Objekt

```
frame.setContentPane(bild.....)
```

```
bild.remove (pic[i]);
```

```
frame.repaint(); // darf bei WINDOWS nicht fehlen
```

```
bild.add (pic[pcnt] =
```

```
    new IMPRImageBean (picLis[pos].picnam, bild, xp, yp, br, ho));
```

```
frame.repaint();
```

die erste frame.repaint() - Anweisung darf bei WINDOWS nicht fehlen

8. LITERATUR

- Java von Kopf bis Fuß, Kathy Sierra & Bert Bates,
O'REILLY ISBN-10 3-89721-448-2
- Java ist auch eine Insel, Christian Ullenboom,
Galileo Computing ISBN 978-3-8362-1146-8
- Das Java Tutorial, Sharon Zahhour . Scott Hommel . Jacob Royal . Isaac
Rabinovitch . Tom Risser . Mark Hoebur,
Addition Wesley ISBN 978-3-8273-2482-5
- Parallele und verteilte Anwendungen in Java, Rainer Oechsle,
Hauser ISBN 978-3-446-40714-5

9. Wünsche

- mehr Networking
- Workshop JAVA/NETBEANS, 1 oder 2 Tage
suche Mitstreiter: Helfer und Teilnehmer
Kosten: ca. 25,- je Tag (Verpflegung eingeschlossen)
- Itanium-PC, klein und leise

DANKE

Q & A

J.END

Matthias Förster ber. Ing. • Moldenhauerweg 40 • 64293 Darmstadt • MFBIDA@arcor.de