Bytecode, Class-Loader und Class-Transformer

Bernd Müller

JUG-Ostfalen

14.8.2014

Mögliche Untertitel

Was ist eine Klasse?
Wie kommt sie in die VM?
Und was macht sie da?

Mögliche Untertitel

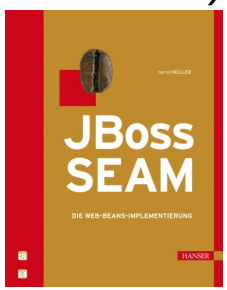
Was ist eine Klasse?
Wie kommt sie in die VM?
Und was macht sie da?

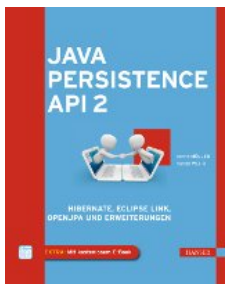
— oder —

Wie man immer alle Unit-Tests besteht ;-)

Vorstellung Referent

- Prof. Informatik (Ostfalia, HS Braunschweig/Wolfenbüttel)
- Buchautor (JSF, Seam, JPA, ...)





- Mitglied EGs JSR 344 (JSF 2.2) und JSR 338 (JPA 2.1)
- Geschäftsführer PMST GmbH
- ...

Motivation

Motivation

- Die schlechte Nachricht:
  - Für Anwendungsentwickler wenig direkt Verwendbares
  - Framework-Entwickler kennen das schon ;-)

Motivation

- Die schlechte Nachricht:
  - Für Anwendungsentwickler wenig direkt Verwendbares
  - Framework-Entwickler kennen das schon ;-)
- Die gute Nachricht:
  - Als interessierter Java-Entwickler lohnt es sich, hinter die Kulissen zu schauen
  - Man versteht einiges besser und wird damit besser
  - Eventuell könnten Sie etwas Spaß haben (siehe Unit-Tests)

Klassen

oder

Wie sieht das binäre Format einer Java-Klasse aus?

C-Struct ClassFile

```
ClassFile {
  u4 magic;
  u2 minor_version;
  u2 major_version;
  u2 constant_pool_count;
  cp_info constant_pool[constant_pool_count-1];
  u2 access_flags;
  u2 this_class;
  u2 super_class;
  u2 interfaces_count;
  u2 interfaces[interfaces_count];
  u2 fields_count;
  field_info fields[fields_count];
  u2 methods_count;
  method_info methods[methods_count];
  u2 attributes_count;
  attribute_info attributes[attributes_count];
}
```

C-Struct `ClassFile` (cont'd)

- `magic`: Magic Number zur Formatidentifizierung, Wert: `0xCAFEBABE`

C-Struct `ClassFile` (cont'd)

- `magic`: Magic Number zur Formatidentifizierung, Wert: `0xCAFEBABE`
- `minor_version`, `major_version`: Versionsnummer für Class-File-Format

1.0	1.1	1.2	1.3	1.4	5.0	6.0	7.0	8.0	9.0 (EA)	...
45.0	45.3	46.0	47.0	48.0	49.0	50.0	51.0	52.0	52.0	...

C-Struct `ClassFile` (cont'd)

- `magic`: Magic Number zur Formatidentifizierung, Wert: `0xCAFEBABE`
- `minor_version`, `major_version`: Versionsnummer für Class-File-Format

1.0	1.1	1.2	1.3	1.4	5.0	6.0	7.0	8.0	9.0 (EA)	...
45.0	45.3	46.0	47.0	48.0	49.0	50.0	51.0	52.0	52.0	...

- `constant_pool count`, `constant_pool[]`: Structs, für String-Literale, Klassen-, Interface-, Field-Namen, etc., die in Klasse referenziert werden

C-Struct `ClassFile` (cont'd)

- `magic`: Magic Number zur Formatidentifizierung, Wert: `0xCAFEBABE`
- `minor_version`, `major_version`: Versionsnummer für Class-File-Format

1.0	1.1	1.2	1.3	1.4	5.0	6.0	7.0	8.0	9.0 (EA)	...
45.0	45.3	46.0	47.0	48.0	49.0	50.0	51.0	52.0	52.0	...

- `constant_pool count`, `constant_pool[]`: Structs, für String-Literale, Klassen-, Interface-, Field-Namen, etc., die in Klasse referenziert werden
- ...

Eine einfache (und sinnlose) Klasse als Beispiel

```
package de.pdbm.simple;
public class Main {

  static final int CONST = 4711;
  private int i;
  private String hello = "Hello World!";

  public Main() {
    i = CONST;
  }
  public static void main(String[] args) {
    Main main = new Main();
    System.out.println(main.hello + main.i);
  }
}
```

Ausgabe von „hexdump -C Main.class“

```
00000000  ca fe ba be 00 00 00 33  00 40 07 00 02 01 00 13  |.......3.@......|
00000010  64 65 2f 70 64 62 6d 2f  73 69 6d 70 6c 65 2f 4d  |de/pdbm/simple/M|
00000020  61 69 6e 07 00 04 01 00  10 6a 61 76 61 2f 6c 61  |ain......java/la|
00000030  6e 67 2f 4f 62 6a 65 63  74 01 00 05 43 4f 4e 53  |ng/Object...CONS|
00000040  54 01 00 01 49 01 00 0d  43 6f 6e 73 74 61 6e 74  |T...I...Constant|
00000050  56 61 6c 75 65 03 00 00  12 67 01 00 01 69 01 00  |Value....g...i..|
00000060  05 68 65 6c 6c 6f 01 00  12 4c 6a 61 76 61 2f 6c  |.hello...Ljava/l|
00000070  61 6e 67 2f 53 74 72 69  6e 67 3b 01 00 06 3c 69  |ang/String;...<i|
00000080  6e 69 74 3e 01 00 03 28  29 56 01 00 04 43 6f 64  |nit>...()V...Cod|
00000090  65 0a 00 03 00 10 0c 00  0c 00 0d 08 00 12 01 00  |e...............|
000000a0  0c 48 65 6c 6c 6f 20 57  6f 72 6c 64 21 09 00 01  |.Hello World!...|
000000b0  00 14 0c 00 0a 00 0b 09  00 01 00 16 0c 00 09 00  |................|
000000c0  06 01 00 0f 4c 69 6e 65  4e 75 6d 62 65 72 54 61  |....LineNumberTa|
000000d0  62 6c 65 01 00 12 4c 6f  63 61 6c 56 61 72 69 61  |ble...LocalVaria|
000000e0  62 6c 65 54 61 62 6c 65  01 00 04 74 68 69 73 01  |bleTable...this.|
000000f0  00 15 4c 64 65 2f 70 64  62 6d 2f 73 69 6d 70 6c  |..Lde/pdbm/simpl|
00000100  65 2f 4d 61 69 6e 3b 01  00 04 6d 61 69 6e 01 00  |e/Main;...main..|
00000110  16 28 5b 4c 6a 61 76 61  2f 6c 61 6e 67 2f 53 74  |.([Ljava/lang/St|
...
```

Ausgabe von „hexdump -C Main.class“

```
00000000  ca fe ba be 00 00 00 33  00 40 07 00 02 01 00 13  |.......3.@......|
00000010  64 65 2f 70 64 62 6d 2f  73 69 6d 70 6c 65 2f 4d  |de/pdbm/simple/M|
00000020  61 69 6e 07 00 04 01 00  10 6a 61 76 61 2f 6c 61  |ain......java/la|
00000030  6e 67 2f 4f 62 6a 65 63  74 01 00 05 43 4f 4e 53  |ng/Object...CONS|
00000040  54 01 00 01 49 01 00 0d  43 6f 6e 73 74 61 6e 74  |T...I...Constant|
00000050  56 61 6c 75 65 03 00 00  12 67 01 00 01 69 01 00  |Value....g...i..|
00000060  05 68 65 6c 6c 6f 01 00  12 4c 6a 61 76 61 2f 6c  |.hello...Ljava/l|
00000070  61 6e 67 2f 53 74 72 69  6e 67 3b 01 00 06 3c 69  |ang/String;...<i|
00000080  6e 69 74 3e 01 00 03 28  29 56 01 00 04 43 6f 64  |nit>...()V...Cod|
00000090  65 0a 00 03 00 10 0c 00  0c 00 0d 08 00 12 01 00  |e...............|
000000a0  0c 48 65 6c 6c 6f 20 57  6f 72 6c 64 21 09 00 01  |.Hello World!...|
000000b0  00 14 0c 00 0a 00 0b 09  00 01 00 16 0c 00 09 00  |................|
000000c0  06 01 00 0f 4c 69 6e 65  4e 75 6d 62 65 72 54 61  |....LineNumberTa|
000000d0  62 6c 65 01 00 12 4c 6f  63 61 6c 56 61 72 69 61  |ble...LocalVaria|
000000e0  62 6c 65 54 61 62 6c 65  01 00 04 74 68 69 73 01  |bleTable...this.|
000000f0  00 15 4c 64 65 2f 70 64  62 6d 2f 73 69 6d 70 6c  |..Lde/pdbm/simpl|
00000100  65 2f 4d 61 69 6e 3b 01  00 04 6d 61 69 6e 01 00  |e/Main;...main..|
00000110  16 28 5b 4c 6a 61 76 61  2f 6c 61 6e 67 2f 53 74  |.([Ljava/lang/St|
...
```

`javap` des SDKs

- `javap` – The Java Class File Disassembler
- **Description**
  The javap command disassembles one or more class files. Its output depends on the options used. If no options are used, javap prints out the package, protected, and public fields and methods of the classes passed to it. javap prints its output to stdout.

Ausgabe von: `javap Main.class`

```
public class de.pdbm.simple.Main {
  static final int CONST;
  public de.pdbm.simple.Main();
  public static void main(java.lang.String[]);
}
```

Ausgabe von: `javap -p Main.class` (private)

```
public class de.pdbm.simple.Main {
  static final int CONST;
  private int i;
  private java.lang.String hello;
  public de.pdbm.simple.Main();
  public static void main(java.lang.String[]);
}
```

Ausgabe von: javap -constants Main.class

```
public class de.pdbm.simple.Main {
  static final int CONST = 4711;
  public de.pdbm.simple.Main();
  public static void main(java.lang.String[]);
}
```

Ausgabe von: `javap -v Main.class` (verbose)

```
Constant pool:
   #1 = Class              #2             //  de/pdbm/simple/Main
   #2 = Utf8               de/pdbm/simple/Main
   #3 = Class              #4             //  java/lang/Object
   #4 = Utf8               java/lang/Object
   #5 = Utf8               CONST
   #6 = Utf8               I
   #7 = Utf8               ConstantValue
   #8 = Integer            4711
   #9 = Utf8               i
  #10 = Utf8               hello
  #11 = Utf8               Ljava/lang/String;
  #12 = Utf8               <init>
  #13 = Utf8               ()V
  #14 = Utf8               Code
  #15 = Methodref          #3.#16         //  java/lang/Object."<init>":()V
  #16 = NameAndType        #12:#13        //  "<init>":()V
  #17 = String             #18            //  Hello World!
  #18 = Utf8               Hello World!
  ...
```

Der Constant Pool

```
cp_info {
  u1 tag;
  u1 info[];
}
```

- Also: Ein Byte für Tag, ein Byte für Info
- Tag beschreibt Struktur für Info

Constant Pool Beispiel

```
#1 = Class              #2                    //  de/pdbm/simple/Ma
#2 = Utf8               de/pdbm/simple/Main
#3 = Class              #4                    //  java/lang/Object
#4 = Utf8               java/lang/Object
```

- 1: Klasse referenziert Utf8 (ein String)
- 2: Utf8 referenziert de/pdbm/simple/Main
- 3: Klasse referenziert Utf8
- 4: Utf8 referenziert java/lang/Object, die Oberklasse

Constant Pool Beispiel (cont)

```
#5 = Utf8               CONST
#6 = Utf8               I
#7 = Utf8               ConstantValue
#8 = Integer            4711
```

- 5: Variable (Name)
- 6: vom Typ Integer
- 7: mit Wert
- 8: 4711

Constant Pool

- Dieses Format des Constant Pools wird beim Laden einer Klasse in die VM verifiziert, d.h. auf Konsistenz geprüft
- Für uns ist es bis hier her ok. Wir wollen das nicht im Detail verstehen
- Sondern uns den interessanteren Dingen zuwenden
- Bemerkung: Das vorgestellte Format ist VM-spezifisch, nicht Java-spezifisch. Andere VM-Sprachen (Groovy, Scala, ...) müssen sich ebenfalls daran halten

Werkzeugunterstützung

- Das Werkzeug `jclasslib` ist ein Byte-Code-Viewer
- Von der Firma EJ-Technologies
- Open-Source unter GPL
- Verfügbar:
  http://www.ej-technologies.com/products/jclasslib/overview.html
- Zusätzlich Bibliothek für den Zugriff auf den Byte-Code

Class-Loader

oder

Wie kommen die Klassen in die VM?

Motivation: Ein Blick hinter die Kulissen

- Class-Loader arbeiten im Hintergrund, daher für Hello-World-Niveau vernachlässigbar

Motivation: Ein Blick hinter die Kulissen

- Class-Loader arbeiten im Hintergrund, daher für Hello-World-Niveau vernachlässigbar
- Für Java 1.1 gezeigt, dass der FQN nicht reicht: *Java is not type-safe*, Vijay Saraswat, AT&T Research, 1997

Motivation: Ein Blick hinter die Kulissen

- Class-Loader arbeiten im Hintergrund, daher für Hello-World-Niveau vernachlässigbar
- Für Java 1.1 gezeigt, dass der FQN nicht reicht: *Java is not type-safe*, Vijay Saraswat, AT&T Research, 1997
- In Java 1.2 neue Class-Loader-Architektur, die seither Bestand hat

Motivation: Ein Blick hinter die Kulissen

- Class-Loader arbeiten im Hintergrund, daher für Hello-World-Niveau vernachlässigbar
- Für Java 1.1 gezeigt, dass der FQN nicht reicht: *Java is not type-safe*, Vijay Saraswat, AT&T Research, 1997
- In Java 1.2 neue Class-Loader-Architektur, die seither Bestand hat
- Einige SDK-Klassen haben einen Parameter vom Typ `ClassLoader` und es gibt Implementierungen der Klasse `ClassLoader`

Motivation: Ein Blick hinter die Kulissen

- Class-Loader arbeiten im Hintergrund, daher für Hello-World-Niveau vernachlässigbar
- Für Java 1.1 gezeigt, dass der FQN nicht reicht: *Java is not type-safe*, Vijay Saraswat, AT&T Research, 1997
- In Java 1.2 neue Class-Loader-Architektur, die seither Bestand hat
- Einige SDK-Klassen haben einen Parameter vom Typ `ClassLoader` und es gibt Implementierungen der Klasse `ClassLoader`
- Class-Loader wichtig in Java-EE. Was sagt die Spezifikation?

Java EE 6 (JSR 316)

- **EE.8.3 Class Loading Requirements**
  The Java EE specification purposely does not define the exact types and arrangements of class loaders that must be used by a Java EE product. Instead, the specification defines requirements in terms of what classes must or must not be visible to components. A Java EE product is free to use whatever class loaders it chooses to meet these requirements. Portable applications must not depend on the types of class loaders used or the hierarchical arrangement of class loaders, if any. Applications should use the techniques described in Section EE.8.2.5, “Dynamic Class Loading” if they need to load classes dynamically.

Class-Loader Basics

- Die VM
  - lädt
  - linkt
  - und initialisiert

  Klassen und Interfaces dynamisch

Class-Loader Basics

- Die VM
  - lädt
  - linkt
  - und initialisiert

  Klassen und Interfaces dynamisch
- Laden: Finden einer Binärdarstellung eines Klassen- oder Interface-Typs eines bestimmten Namen und *Erzeugen* der Klasse/Interface aus dieser Binärdarstellung

Class-Loader Basics

- Die VM
  - lädt
  - linkt
  - und initialisiert

  Klassen und Interfaces dynamisch
- Laden: Finden einer Binärdarstellung eines Klassen- oder Interface-Typs eines bestimmten Namen und *Erzeugen* der Klasse/Interface aus dieser Binärdarstellung
- Linking: Einfügen der Klasse/Interface in den aktuellen Laufzeitzustand der VM, so dass sie verwendet werden kann

Class-Loader Basics

- Die VM
  - lädt
  - linkt
  - und initialisiert

  Klassen und Interfaces dynamisch
- Laden: Finden einer Binärdarstellung eines Klassen- oder Interface-Typs eines bestimmten Namen und *Erzeugen* der Klasse/Interface aus dieser Binärdarstellung
- Linking: Einfügen der Klasse/Interface in den aktuellen Laufzeitzustand der VM, so dass sie verwendet werden kann
- Initialisierung: Ausführen der Initialisierungsmethode `<clinit>`

Laden und Linken

- Laden erzeugt (mit ein paar grundlegenden Prüfungen) ein `Class`-Objekt, das aber noch nicht verwendet werden kann

Laden und Linken

- Laden erzeugt (mit ein paar grundlegenden Prüfungen) ein `Class`-Objekt, das aber noch nicht verwendet werden kann
- Linken besteht aus weiteren einzelnen Schritten:
  - Verification
  - Preparation
  - Resolution

Verification

- Die Verifikation bestätigt, dass sich die Klasse „gutmütig“ verhält und keine Laufzeitprobleme verursacht

Verification

- Die Verifikation bestätigt, dass sich die Klasse „gutmütig“ verhält und keine Laufzeitprobleme verursacht
- Einige dieser Prüfungen:
  - Constant Pool konsistent
  - Keine final-Methoden überschrieben
  - Methoden respektieren Access-Control
  - Methoden werden mit korrekter Anzahl und Typ von Parametern aufgerufen
  - Byte-Code manipuliert Stack nicht
  - Variablen sind vor Verwendung initialisiert
  - Variablen werden nur Werte passender Typen zugewiesen
  - ...

Verification

- Die Verifikation bestätigt, dass sich die Klasse „gutmütig“ verhält und keine Laufzeitprobleme verursacht
- Einige dieser Prüfungen:
  - Constant Pool konsistent
  - Keine final-Methoden überschrieben
  - Methoden respektieren Access-Control
  - Methoden werden mit korrekter Anzahl und Typ von Parametern aufgerufen
  - Byte-Code manipuliert Stack nicht
  - Variablen sind vor Verwendung initialisiert
  - Variablen werden nur Werte passender Typen zugewiesen
  - ...
- Bei Verletzung wird `VerifyError` geworfen

Preparation und Resolution

- Preparation:
  Vorbereitung, dass Speicher für Klasse alloziert und Klassenvariablen initialisiert werden können. Initalisierung wird aber nicht durchgeführt und es wird kein Byte-Code ausgeführt

Preparation und Resolution

- Preparation:
  Vorbereitung, dass Speicher für Klasse alloziert und Klassenvariablen initialisiert werden können. Initalisierung wird aber nicht durchgeführt und es wird kein Byte-Code ausgeführt
- Resolution:
  Prüft, dass alle von der Klasse referenzierten Klassen geladen sind. Falls nicht, werden diese geladen

Preparation und Resolution

- Preparation:
  Vorbereitung, dass Speicher für Klasse alloziert und Klassenvariablen initialisiert werden können. Initalisierung wird aber nicht durchgeführt und es wird kein Byte-Code ausgeführt
- Resolution:
  Prüft, dass alle von der Klasse referenzierten Klassen geladen sind. Falls nicht, werden diese geladen
- Wenn alle Klassen geladen sind, wird Initialisierung (Klassenvariablen und Static-Initializer-Blöcke) durchgeführt

Preparation und Resolution

- Preparation:
  Vorbereitung, dass Speicher für Klasse alloziert und Klassenvariablen initialisiert werden können. Initalisierung wird aber nicht durchgeführt und es wird kein Byte-Code ausgeführt
- Resolution:
  Prüft, dass alle von der Klasse referenzierten Klassen geladen sind. Falls nicht, werden diese geladen
- Wenn alle Klassen geladen sind, wird Initialisierung (Klassenvariablen und Static-Initializer-Blöcke) durchgeführt
- Wenn das abgeschlossen ist, steht die Klasse zur Verwendung bereit

API-Doc `ClassLoader`

```
public abstract class ClassLoader extends Object
```

A class loader is an object that is responsible for loading classes. The class `ClassLoader` is an abstract class. . . .

Every Class object contains a reference to the ClassLoader that defined it. . . .

Applications implement subclasses of ClassLoader in order to extend the manner in which the Java virtual machine dynamically loads classes.

Class loaders may typically be used by security managers to indicate security domains.

API-Doc `ClassLoader` (cont'd)

The ClassLoader class uses a delegation model to search for classes and resources. Each instance of ClassLoader has an associated parent class loader. When requested to find a class or resource, a ClassLoader instance will delegate the search for the class or resource to its parent class loader before attempting to find the class or resource itself. The virtual machine's built-in class loader, called the "bootstrap class loader", does not itself have a parent but may serve as the parent of a ClassLoader instance.

Class loaders that support concurrent loading of classes are known as parallel capable class loaders and are required to register themselves at their class initialization time by invoking the ClassLoader.registerAsParallelCapable method. Note that the ClassLoader class is registered as parallel capable by default. However, its subclasses still need to register themselves if they are parallel capable. In environments in which the delegation model is not strictly hierarchical, class loaders need to be parallel capable,

...

Class-Loader-Basics zum Zweiten

- Henne-Ei-Problem: Erzeugung einer Class-Loader-Instanz setzt voraus, dass dieser Class-Loader bereits geladen wurde. Von wem?

Class-Loader-Basics zum Zweiten

- Henne-Ei-Problem: Erzeugung einer Class-Loader-Instanz setzt voraus, dass dieser Class-Loader bereits geladen wurde. Von wem?
- Der Class-Loader ist eine normale Klasse und leitet von Object ab. Object muss also geladen worden sein. Von wem?

Class-Loader-Basics zum Zweiten

- Henne-Ei-Problem: Erzeugung einer Class-Loader-Instanz setzt voraus, dass dieser Class-Loader bereits geladen wurde. Von wem?
- Der Class-Loader ist eine normale Klasse und leitet von Object ab. Object muss also geladen worden sein. Von wem?
- Im API-Doc schon genannt: der Bootstrap-Class-Loader auch Primordinal Class-Loader genannt

Class-Loader-Basics zum Zweiten

- Henne-Ei-Problem: Erzeugung einer Class-Loader-Instanz setzt voraus, dass dieser Class-Loader bereits geladen wurde. Von wem?
- Der Class-Loader ist eine normale Klasse und leitet von Object ab. Object muss also geladen worden sein. Von wem?
- Im API-Doc schon genannt: der Bootstrap-Class-Loader auch Primordinal Class-Loader genannt
- Und durch das Delegationsmodell ergibt sich eine Hierarchie: Alle Class-Loader haben einen Parent außer Bootstrap-Class-Loader

Class-Loader-Hierarchie

- **Bootstrap-Class-Loader**
  - Sehr früh bei VM-Start instanziiert
  - Meist nativ implementiert
  - Gehört praktisch zur VM
  - Lädt System-JARs, z.B. `rt.jar`
  - Verifiziert nicht
  - Path-Property: `sun.boot.class.path`

Class-Loader-Hierarchie

- **Bootstrap-Class-Loader**
  - Sehr früh bei VM-Start instanziiert
  - Meist nativ implementiert
  - Gehört praktisch zur VM
  - Lädt System-JARs, z.B. `rt.jar`
  - Verifiziert nicht
  - Path-Property: `sun.boot.class.path`
- **Extension-Class-Loader**
  - Lädt Standarderweiterungen
  - Path-Property: `java.ext.dirs`
  - Klasse: `sun.misc.Launcher$ExtClassLoader`

Class-Loader-Hierarchie (cont'd)

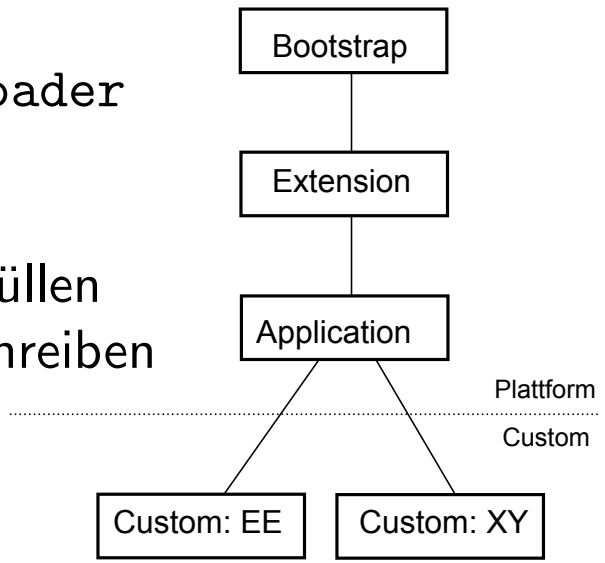
- **Application-Class-Loader**
  - Lädt Anwendungsklassen
  - In SE der Class-Loader, der die meisten Klassen lädt
  - Path-Property: `java.class.path`
  - Klasse: `sun.misc.Launcher$AppClassLoader`

Class-Loader-Hierarchie (cont'd)

- **Application-Class-Loader**
  - Lädt Anwendungsklassen
  - In SE der Class-Loader, der die meisten Klassen lädt
  - Path-Property: `java.class.path`
  - Klasse: `sun.misc.Launcher$AppClassLoader`
- **Custom-Class-Loader**
  - Wird in Java-EE benötigt, um Spec zu erfüllen
  - Jeder (Sie?) kann eigenen Class-Loader schreiben
  - Bekanntes Beispiel: JBoss Modules

Class-Loader-Hierarchie (cont'd)

- **Application-Class-Loader**
  - Lädt Anwendungsklassen
  - In SE der Class-Loader, der die meisten Klassen lädt
  - Path-Property: `java.class.path`
  - Klasse: `sun.misc.Launcher$AppClassLoader`
- **Custom-Class-Loader**
  - Wird in Java-EE benötigt, um Spec zu erfüllen
  - Jeder (Sie?) kann eigenen Class-Loader schreiben
  - Bekanntes Beispiel: JBoss Modules



Kann man das überprüfen?

- Class-Loader-Option der VM: -verbose:class

```
  1: [Opened /usr/lib/jvm/.../jre/lib/rt.jar]
  2: [Loaded java.lang.Object from /usr/lib/jvm/.../jre/lib/rt.jar]
  3: [Loaded java.io.Serializable from /usr/lib/jvm/.../jre/lib/rt.jar]
 12: [Loaded java.lang.ClassLoader from /usr/lib/jvm/.../jre/lib/rt.jar
 55: [Loaded sun.reflect.DelegatingClassLoader from /usr/lib/jvm/.../jr
223: [Loaded java.lang.ClassLoader$3 from /usr/lib/jvm/.../jre/lib/rt.j
228: [Loaded java.lang.ClassLoader$NativeLibrary from /usr/lib/jvm/.../
243: [Loaded java.security.SecureClassLoader from /usr/lib/jvm/.../jre/
244: [Loaded java.net.URLClassLoader from /usr/lib/jvm/.../jre/lib/rt.j
245: [Loaded sun.misc.Launcher$ExtClassLoader from /usr/lib/jvm/.../jre
247: [Loaded java.lang.ClassLoader$ParallelLoaders from /usr/lib/jvm/ja
256: [Loaded java.net.URLClassLoader$7 from /usr/lib/jvm/java-1.7.0-ope
259: [Loaded sun.misc.Launcher$ExtClassLoader$1 from /usr/lib/jvm/java-
317: [Loaded sun.misc.Launcher$AppClassLoader from /usr/lib/jvm/java-1.
318: [Loaded sun.misc.Launcher$AppClassLoader$1 from /usr/lib/jvm/java-
319: [Loaded java.lang.SystemClassLoaderAction from /usr/lib/jvm/java-1
321: [Loaded java.net.URLClassLoader$1 from /usr/lib/jvm/java-1.7.0-ope
383: [Loaded de.pdbm.simple.Main from file:/home/bernd/lehre/skripte/cl
384: [Loaded java.lang.Void from /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.
385: Hello World!4711
```

Class-Loader-Delegation

- `ClassLoader` ist Oberklasse aller Class-Loader

Class-Loader-Delegation

- `ClassLoader` ist Oberklasse aller Class-Loader
- `ClassLoader` enthält:

```
private final ClassLoader parent;
```

Class-Loader-Delegation

- `ClassLoader` ist Oberklasse aller Class-Loader
- `ClassLoader` enthält:

  ```
  private final ClassLoader parent;
  ```
- Ganz nett: Kommentar aus src.zip:

  ```
  // The parent class loader for delegation
  // Note: VM hardcoded the offset of this field, thus all
  // new fields must be added *after* it.
  ```

Class-Loader-Delegation

- `ClassLoader` ist Oberklasse aller Class-Loader
- `ClassLoader` enthält:
  `private final ClassLoader parent;`
- Ganz nett: Kommentar aus src.zip:

```
// The parent class loader for delegation
// Note: VM hardcoded the offset of this field, thus all
// new fields must be added *after* it.
```

- Und Konstruktor für Delegation an Parent:
  `protected ClassLoader(ClassLoader parent)`

Class-Loader-Delegation

- `ClassLoader` ist Oberklasse aller Class-Loader
- `ClassLoader` enthält:
  ```
  private final ClassLoader parent;
  ```
- Ganz nett: Kommentar aus src.zip:
  ```
  // The parent class loader for delegation
  // Note: VM hardcoded the offset of this field, thus all
  // new fields must be added *after* it.
  ```
- Und Konstruktor für Delegation an Parent:
  ```
  protected ClassLoader(ClassLoader parent)
  ```
- Und Methode
  ```
  protected Class<?> loadClass(String name,
                               boolean resolve)
  ```

Class-Loader-Delegation

- `ClassLoader` ist Oberklasse aller Class-Loader
- `ClassLoader` enthält:
  ```
  private final ClassLoader parent;
  ```
- Ganz nett: Kommentar aus src.zip:
  ```
  // The parent class loader for delegation
  // Note: VM hardcoded the offset of this field, thus all
  // new fields must be added *after* it.
  ```
- Und Konstruktor für Delegation an Parent:
  ```
  protected ClassLoader(ClassLoader parent)
  ```
- Und Methode
  ```
  protected Class<?> loadClass(String name,
                               boolean resolve)
  ```
- Falls Parent `null` Bootstrap-Class-Loader

API-Doc der Methode `loadClass()`

Loads the class with the specified binary name. The default implementation of this method searches for classes in the following order:

1. Invoke findLoadedClass(String) to check if the class has already been loaded.
2. Invoke the loadClass method on the parent class loader. If the parent is null the class loader built-in to the virtual machine is used, instead.
3. Invoke the findClass(String) method to find the class.

If the class was found using the above steps, and the resolve flag is true, this method will then invoke the resolveClass(Class) method on the resulting Class object.

API-Doc der Methode `loadClass()`

Loads the class with the specified binary name. The default implementation of this method searches for classes in the following order:

1. Invoke findLoadedClass(String) to check if the class has already been loaded.
2. Invoke the loadClass method on the parent class loader. If the parent is null the class loader built-in to the virtual machine is used, instead.
3. Invoke the findClass(String) method to find the class.

If the class was found using the above steps, and the resolve flag is true, this method will then invoke the resolveClass(Class) method on the resulting Class object.

- Nennt man *Parent-First*-Strategie

Delegation an AppClassLoader

```
public class Different { main(){

URL url = new URL("file://.");
URLClassLoader loader1 = new URLClassLoader(new URL[]{url})
URLClassLoader loader2 = new URLClassLoader(new URL[]{url})
System.out.println(loader1.equals(loader2)); // false
System.out.println(loader1.getParent());     // AppClassLoa
System.out.println(loader2.getParent());     // AppClassLoa
Class<?> class1 = loader1.loadClass(CLASS);
System.out.println(Different.class.equals(class1)); //false
Class<?> class2 = loader2.loadClass(CLASS);
System.out.println(class2.equals(class1));   // true
System.out.println(class1.getClassLoader()); // AppClassLoa
System.out.println(class2.getClassLoader()); // AppClassLoa
```

Keine Delegation

```
public class Different { main(){

URL url = new URL("file://.");
URLClassLoader loader1 =
    new URLClassLoader(new URL[]{url}, null);
URLClassLoader loader2 =
    new URLClassLoader(new URL[]{url}, null);
System.out.println(loader1.equals(loader2)); // false
System.out.println(loader1.getParent());     // null
System.out.println(loader2.getParent());     // null
Class<?> class1 = loader1.loadClass(CLASS);  // CNFE
...
```

Beispiel Hierarchie

```
public class Hierarchy {

 public static void main(String[] args) {
   ClassLoader classLoader = Hierarchy.class.getClassLoader
   do
     System.out.println(classLoader);
   while ((classLoader = classLoader.getParent()) != null);
   System.out.println(Object.class.getClassLoader());
 }
}
```

- Ausgabe:

```
sun.misc.Launcher$AppClassLoader@2ed4a1d3
sun.misc.Launcher$ExtClassLoader@9cc3baa
null
```

Thread-Context-Class-Loader

- Parent-First-Strategie nicht immer möglich

Thread-Context-Class-Loader

- Parent-First-Strategie nicht immer möglich
- Beispiel: JAXP-Fabriken werden vom Bootstrap-Class-Loader geladen, da in rt.jar. Alternative Implementierungen werden aber von App-Class-Loader geladen

Thread-Context-Class-Loader

- Parent-First-Strategie nicht immer möglich
- Beispiel: JAXP-Fabriken werden vom Bootstrap-Class-Loader geladen, da in rt.jar. Alternative Implementierungen werden aber von App-Class-Loader geladen
- Damit muss das Laden von einem Parent- an einen Child-Loader delegiert werden, was nicht geht

Thread-Context-Class-Loader

- Parent-First-Strategie nicht immer möglich
- Beispiel: JAXP-Fabriken werden vom Bootstrap-Class-Loader geladen, da in rt.jar. Alternative Implementierungen werden aber von App-Class-Loader geladen
- Damit muss das Laden von einem Parent- an einen Child-Loader delegiert werden, was nicht geht
- Lösung: jedem Thread wird ein Class-Loader zugeordnet (seit Java 1.2), der nicht parent-first ist, der sogenannte Thread-Context-Class-Loader

Thread-Context-Class-Loader

- Parent-First-Strategie nicht immer möglich
- Beispiel: JAXP-Fabriken werden vom Bootstrap-Class-Loader geladen, da in rt.jar. Alternative Implementierungen werden aber von App-Class-Loader geladen
- Damit muss das Laden von einem Parent- an einen Child-Loader delegiert werden, was nicht geht
- Lösung: jedem Thread wird ein Class-Loader zugeordnet (seit Java 1.2), der nicht parent-first ist, der sogenannte Thread-Context-Class-Loader

▸ Abkürzung

Beispiel 1: Thread-Context-Class-Loader

```
public class TCCL {
  public static void main(String[] args) {
    ClassLoader tccl = Thread.currentThread()
                                .getContextClassLoader();
    ClassLoader cl = TCCL.class.getClassLoader();
    ClassLoader scl = ClassLoader.getSystemClassLoader();
    System.out.println("TCCL: " + tccl);
    System.out.println("CL: " + cl);
    System.out.println("SCL: " + scl);
  }
}
```

- Ausgabe:

```
TCCL: sun.misc.Launcher$AppClassLoader@9cc3baa
CL: sun.misc.Launcher$AppClassLoader@9cc3baa
SCL: sun.misc.Launcher$AppClassLoader@9cc3baa
```

Beispiel 2: Thread-Context-Class-Loader

```
public class TCCL {
  public static void main(String[] args) {
    SAXParserFactory saxParserFactory = SAXParserFactory
                                        .newInstance();
    System.out.println(saxParserFactory.getClass());
    System.out.println(saxParserFactory.getClass()
                                       .getClassLoader());
  }
}
```

- Ausgabe:

```
class com.sun.org.apache.xerces.internal.jaxp.SAXParserFa
null
```

Beispiel 3: Thread-Context-Class-Loader

- Gleiches Programm, aber Aufruf mit separatem Xerces

```
java -cp /java/libs/xercesImpl.jar:.
     -Djavax.xml.perser.SAXParserFactory=\
       org.apache.xerces.jaxp.SaxParserFactoryImpl
     de.pdbm.classloader.TCCL
```

- Ausgabe:

```
class org.apache.xerces.jaxp.SAXParserFactoryImpl
sun.misc.Launcher$AppClassLoader@645ad7b2
```

TCCL intern aber wieder Parent-First

- TCCL nutzt intern Parent-First-Strategie, erlaubt aber das Setzen des Class-Loaders
- Klasse `Thread` mit Methoden
  - `getContextClassLoader()`
  - `setContextClassLoader(ClassLoader)`

The Java 2 platform also introduced the notion of context class loader. A thread's context class loader is, by default, set to the context class loader of the thread's parent. The hierarchy of threads is rooted at the primordial thread (the one that runs the program). The context class loader of the primordial thread is set to the class loader that loaded the application. So unless you explicitly change the thread's context class loader, its context class loader will be the application's class loader. That is, the context class loader can load the classes that the application can load. This loader is used by the Java runtime such as the RMI (Java Remote Method Invocation) to load classes and resources on behalf of the user application. The context class loader, like any Java 2 platform class loader, has a parent class loader and supports the same delegation model for class loading described previously. (SE Tutorial, Class Loading)

Kleines Quiz

- Was passiert, wenn zur Laufzeit `Missing.class` fehlt?

```
public class Strange1 {
  public static void main(String[] args) {
    try {
      Missing m = new Missing();
    } catch (java.lang.NoClassDefFoundError e) {
      ...
    }
  }
}
public class Strange2 {
  public static void main(String[] args) {
    Missing m;
    try {
      m = new Missing();
    } catch (java.lang.NoClassDefFoundError e) {
      ...
    }
  }
}
```

Randnotiz: Implementierungsabhängigkeiten

Java	1.4.2	5	6	7	8
Strange1	nicht abg.	nicht abg.	abgef.	abgef.	abgef.
Strange2	abgef.	abgef.	abgef.	abgef.	abgef.

Randnotiz: Implementierungsabhängigkeiten

Java	1.4.2	5	6	7	8
Strange1	nicht abg.	nicht abg.	abgef.	abgef.	abgef.
Strange2	abgef.	abgef.	abgef.	abgef.	abgef.

- JLS 12.2.1 The Loading Process
- „If an error occurs during class loading, then an instance of one of the following subclasses of class `LinkageError` will be thrown at any point in the program that (directly or indirectly) uses the type.“
- `NoClassDefFoundError` ist eine dieser Unterklassen.
- Es gibt also keine konkrete Aussage. Das Entstehen der Exception ist *implementierungsabhängig*

Pre-Main

oder

Gibt es ein Leben vor `main()` ?

Das Package `java.lang.instrument`

- Das Package `java.lang.instrument` wurde mit Java 5 eingeführt

Das Package `java.lang.instrument`

- Das Package `java.lang.instrument` wurde mit Java 5 eingeführt
- Aus dem API-Doc:
  „*Provides services that allow Java programming language agents to instrument programs running on the JVM. The mechanism for instrumentation is modification of the byte-codes of methods.*“

Das Package `java.lang.instrument`

- Das Package `java.lang.instrument` wurde mit Java 5 eingeführt
- Aus dem API-Doc:
  „*Provides services that allow Java programming language agents to instrument programs running on the JVM. The mechanism for instrumentation is modification of the byte-codes of methods.*“
- Wikipedia *Instrumentation* (computer programming):
  „*. . . instrumentation refers to an ability to monitor or measure the level of a product's performance, to diagnose errors and to write trace information. . . .*“

Das Package `java.lang.instrument`

- Das Package `java.lang.instrument` wurde mit Java 5 eingeführt
- Aus dem API-Doc:
  „*Provides services that allow Java programming language agents to instrument programs running on the JVM. The mechanism for instrumentation is modification of the byte-codes of methods.*“
- Wikipedia *Instrumentation* (computer programming):
  „*. . . instrumentation refers to an ability to monitor or measure the level of a product's performance, to diagnose errors and to write trace information. . . .*“
- Also (nach Wikipedia) gedacht für's Monitoring

Das Package `java.lang.instrument`

- Das Package `java.lang.instrument` wurde mit Java 5 eingeführt
- Aus dem API-Doc:
  „*Provides services that allow Java programming language agents to instrument programs running on the JVM. The mechanism for instrumentation is modification of the byte-codes of methods.*“
- Wikipedia *Instrumentation* (computer programming):
  „*. . . instrumentation refers to an ability to monitor or measure the level of a product's performance, to diagnose errors and to write trace information. . . .*“
- Also (nach Wikipedia) gedacht für's Monitoring
- Lässt sich aber für beliebiges nutzen (z.B. von JPA-Providern)

Package Specification `java.lang.instrument`

„*An agent is deployed as a JAR file. An attribute in the JAR file manifest specifies the agent class which will be loaded to start the agent. For implementations that support a command-line interface, an agent is started by specifying an option on the command-line. Implementations may also support a mechanism to start agents some time after the VM has started. For example, an implementation may provide a mechanism that allows a tool to attach to a running application, and initiate the loading of the tool's agent into the running application. The details as to how the load is initiated, is implementation dependent.*“

- Zentral: der Agent

Der Agent

- Deployed als Jar-File

Der Agent

- Deployed als Jar-File
- Attribut im Manifest definiert die Agent-Klasse

Der Agent

- Deployed als Jar-File
- Attribut im Manifest definiert die Agent-Klasse
- Alternativen, um Agent zu starten

Der Agent

- Deployed als Jar-File
- Attribut im Manifest definiert die Agent-Klasse
- Alternativen, um Agent zu starten
  - Auf Kommandozeile bei VM-Start (zwingend erforderlich für Kommandozeilenimplementierungen)

Der Agent

- Deployed als Jar-File
- Attribut im Manifest definiert die Agent-Klasse
- Alternativen, um Agent zu starten
  - Auf Kommandozeile bei VM-Start (zwingend erforderlich für Kommandozeilenimplementierungen)
  - Nach VM-Start, z.B. durch nicht näher spezifiziertes Binden (optional und implementation dependent)

Agentenstart über Kommandozeile

- Syntax: `-javaagent:`*jarpath[=options]*

Agentenstart über Kommandozeile

- Syntax: `-javaagent:`*jarpath[=options]*
- Option mehrfach verwendbar, damit mehrere Agenten

Agentenstart über Kommandozeile

- Syntax: `-javaagent:`*jarpath[=options]*
- Option mehrfach verwendbar, damit mehrere Agenten
- Manifest des Agenten-Jars muss Attribut `Premain-Class` enthalten

Agentenstart über Kommandozeile

- Syntax: `-javaagent:`*jarpath[=options]*
- Option mehrfach verwendbar, damit mehrere Agenten
- Manifest des Agenten-Jars muss Attribut `Premain-Class` enthalten
- Diese Agentenklasse muss `premain()`-Methode enthalten

Agentenstart über Kommandozeile

- Syntax: `-javaagent:`*jarpath[=options]*
- Option mehrfach verwendbar, damit mehrere Agenten
- Manifest des Agenten-Jars muss Attribut `Premain-Class` enthalten
- Diese Agentenklasse muss `premain()`-Methode enthalten
- Nachdem VM initialisiert ist, werden alle `premain()`-Methoden in der Reihenfolge der Optionen aufgerufen, dann die `main()`-Methode

Agentenstart über Kommandozeile

- Syntax: `-javaagent:`*jarpath[=options]*
- Option mehrfach verwendbar, damit mehrere Agenten
- Manifest des Agenten-Jars muss Attribut `Premain-Class` enthalten
- Diese Agentenklasse muss `premain()`-Methode enthalten
- Nachdem VM initialisiert ist, werden alle `premain()`-Methoden in der Reihenfolge der Optionen aufgerufen, dann die `main()`-Methode
- Zwei mögliche Signaturen:

```
public static void premain(String agentArgs,
                           Instrumentation inst);
public static void premain(String agentArgs);
```

- Aufruf der zweiten Alternative nur, falls erste nicht existiert

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start
- Falls Start über Kommandozeile, wird `agentmain()` nicht aufgerufen

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start
- Falls Start über Kommandozeile, wird `agentmain()` nicht aufgerufen
- Agent wird über System-Class-Loader geladen

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start
- Falls Start über Kommandozeile, wird `agentmain()` nicht aufgerufen
- Agent wird über System-Class-Loader geladen
- Jeder Agent bekommt seine Parameter über `agentArgs`-Parameter als *ein* String, d.h. Agent muss selbst parsen

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start
- Falls Start über Kommandozeile, wird `agentmain()` nicht aufgerufen
- Agent wird über System-Class-Loader geladen
- Jeder Agent bekommt seine Parameter über `agentArgs`-Parameter als *ein* String, d.h. Agent muss selbst parsen
- Falls Agent nicht geladen werden kann oder `premain()`-Methode nicht existert, wird VM beendet

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start
- Falls Start über Kommandozeile, wird `agentmain()` nicht aufgerufen
- Agent wird über System-Class-Loader geladen
- Jeder Agent bekommt seine Parameter über `agentArgs`-Parameter als *ein* String, d.h. Agent muss selbst parsen
- Falls Agent nicht geladen werden kann oder `premain()`-Methode nicht existert, wird VM beendet
- Exceptions der `premain()`-Methode führen ebenfalls zum Beenden der VM

Was kann man damit machen?

- Wie bereits erwähnt: Instrumentieren

Was kann man damit machen?

- Wie bereits erwähnt: Instrumentieren
- Z.B. um

Was kann man damit machen?

- Wie bereits erwähnt: Instrumentieren
- Z.B. um
  - Zu monitoren

Was kann man damit machen?

- Wie bereits erwähnt: Instrumentieren
- Z.B. um
  - Zu monitoren
  - Proxies zu bauen (JPA: Assoziationen, Automatic Dirty Checking,. . .)

Was kann man damit machen?

- Wie bereits erwähnt: Instrumentieren
- Z.B. um
  - Zu monitoren
  - Proxies zu bauen (JPA: Assoziationen, Automatic Dirty Checking,...)
  - `final` Modifier entfernen

Was kann man damit machen?

- Wie bereits erwähnt: Instrumentieren
- Z.B. um
  - Zu monitoren
  - Proxies zu bauen (JPA: Assoziationen, Automatic Dirty Checking,...)
  - `final` Modifier entfernen
  - ...

Was kann man damit machen?

- Wie bereits erwähnt: Instrumentieren
- Z.B. um
  - Zu monitoren
  - Proxies zu bauen (JPA: Assoziationen, Automatic Dirty Checking,...)
  - `final` Modifier entfernen
  - ...
  - Allgemein: Sinnvolles Verhalten, das nicht im Code steht, nachträglich und nur bei Bedarf einbauen

Beispiel Monitoring: Aufrufhäufigkeit der Methode

```
public class ClassToMonitor {

  public void foo() {
    // beliebig
  }

}
```

- Aufrufhäufigkeit der Methode `foo()` soll (auf einfache Art) gezählt werden

Beispiel Monitoring: Zähler und Main

```
public class Monitor {
  public static int counter = 0;
}

public class Main {
  public static void main(String[] args)
          throws Exception {
    System.out.println("Zaehler vor Schleife: "
                       + Monitor.counter);
    ClassToMonitor classToMonitor = new ClassToMonitor();
    for (int i = 0; i < 1000; i++) {
      classToMonitor.foo();
    }
    System.out.println("Zaehler nach Schleife: "
                       + Monitor.counter);
  }
}
```

Beispiel Monitoring: Der Agent

```
public class MonitorAgent {

  public static void premain(String agentArgs,
                             Instrumentation instrumentation) {
    instrumentation
        .addTransformer(new MonitorTransformer());
  }
}
```

Und die MANIFEST.MF

```
Premain-Class: de.pdbm.MonitorAgent
```

Beispiel Monitoring: Instrumentierung mit Javassist

```
public class MonitorTransformer
             implements ClassFileTransformer {

  public byte[] transform(ClassLoader loader, String classNa
      Class<?> classBeingRedefined, ProtectionDomain protec
      byte[] classfileBuffer) throws IllegalClassFormatExce

    if (className.equals("de/pdbm/ClassToMonitor")) {
      ClassPool pool = ClassPool.getDefault();
      try {
        CtClass cc = pool.get("de.pdbm.ClassToMonitor");
        CtMethod method = cc.getDeclaredMethod("foo");
        method.insertBefore("de.pdbm.Monitor.counter++;");
        return cc.toBytecode();
      } catch (NotFoundException | CannotCompileException |
        ...
      }
    }
    return classfileBuffer; // andere Klassen unveraendert
  }
}
```

Das Attach-API

oder

Wie rede ich mit einer VM ?

Wiederholung Instrumentation-Package

„. . . Implementations may also support a mechanism to start agents some time after the VM has started. For example, an implementation may provide a mechanism that allows a tool to attach to a running application, and initiate the loading of the tool's agent into the running application. The details as to how the load is initiated, is implementation dependent.“

- Achtung: implementierungsabhängig
- Aber: in HotSpot, JRockit, IBM-VM vorhanden
- Schnittstelle ist die abstrakte Klasse `VirtualMachine` im Package `com.sun.tools.attach`, enthalten in `tools.jar`

API-Doc Klasse `VirtualMachine`

„ *A VirtualMachine represents a Java virtual machine to which this Java virtual machine has attached. The Java virtual machine to which it is attached is sometimes called the target virtual machine, or target VM. An application (typically a tool such as a managemet console or profiler) uses a VirtualMachine to load an agent into the target VM. For example, a profiler tool written in the Java Language might attach to a running application and load its profiler agent to profile the running application.* “

- Methode `attach(<pid>)` Fabrikmethode, um angebundene Instanz zu bekommen
- Methode `loadAgent(<agent>,<args>)`, um Agent zu laden und zu starten (Methode `agentmain()`)

Agent mit agentmain()

- Der Agent

```
public static void agentmain(String agentArgs,
                             Instrumentation inst);
public static void agentmain(String agentArgs);
```

- Im Manifest Attribut `Agent-Class` auf Klassennamen des Agenten setzten
- Attribut `Can-Redefine-Classes` auf `true`, falls redefiniert (neue Klassendefinition)
- Attribut `Can-Retransform-Classes` auf `true`, falls transformiert (Byte-Code-Enhancer)

Beispiel Ändern einer Methode und Neuladen

```
public class ClassToBeRedefined {

  public void saySomething() {
    System.out.println("foo");
    //System.out.println("bar");
  }

}
```

Beispiel Ändern einer Methode und Neuladen (cont'd)

```
public class Agent {

  private static Instrumentation instrumentation = null;

  public static void agentmain(String agentArgument,
                               Instrumentation instrumentation) {
    Agent.instrumentation = instrumentation;
  }

  public static void redefineClasses(ClassDefinition...
                                     definitions) throws Ex
    if (Agent.instrumentation == null) {
      throw new RuntimeException("Agent nicht gestartet. In
    }
    Agent.instrumentation.redefineClasses(definitions);
  }

}
```

Beispiel Ändern einer Methode und Neuladen (cont'd)

```
public class Main {

  public static void main(String[] args) throws Exception {
    ClassToBeRedefined ctbr = new ClassToBeRedefined();
    ctbr.saySomething();
    InputStream is = ctbr.getClass().getClassLoader()
                         // class ClassToBeRedefined
                         .getResourceAsStream("dummy");
    byte[] classBytes = classInputStreamToByteArray(is);
    ClassDefinition classDefinition =
        new ClassDefinition(ctbr.getClass(), classBytes);
    loadAgent();
    Agent.redefineClasses(classDefinition);
    ctbr.saySomething();
  }
  ...
```

Beispiel Ändern einer Methode und Neuladen (cont'd)

```
    ...
    private static void loadAgent() {
      String nameOfRunningVM = ManagementFactory
                               .getRuntimeMXBean().getName();
      int p = nameOfRunningVM.indexOf('@');
      String pid = nameOfRunningVM.substring(0, p);

      try {
        VirtualMachine vm = VirtualMachine.attach(pid);
        vm.loadAgent(JAR_FILE_PATH, "");
        vm.detach();
      } catch (Exception e) {
        throw new RuntimeException(e);
      }
    }
}
```

Auszug aus API-Doc für `redefineClasses()` I

- This method is used to replace the definition of a class without reference to the existing class file bytes, as one might do when recompiling from source for fix-and-continue debugging. Where the existing class file bytes are to be transformed (for example in bytecode instrumentation) retransformClasses should be used.
- This method operates on a set in order to allow interdependent changes to more than one class at the same time (a redefinition of class A can require a redefinition of class B).

Auszug aus API-Doc für `redefineClasses()` II

- If a redefined method has active stack frames, those active frames continue to run the bytecodes of the original method. The redefined method will be used on new invokes.
- This method does not cause any initialization except that which would occur under the customary JVM semantics. In other words, redefining a class does not cause its initializers to be run. The values of static variables will remain as they were prior to the call.
- Instances of the redefined class are not affected.

Auszug aus API-Doc für `redefineClasses()` III

- The redefinition may change method bodies, the constant pool and attributes. The redefinition must not add, remove or rename fields or methods, change the signatures of methods, or change inheritance. These restrictions maybe be lifted in future versions. The class file bytes are not checked, verified and installed until after the transformations have been applied, if the resultant bytes are in error this method will throw an exception.

Retransformation

- Retransformation auch möglich

Retransformation

- Retransformation auch möglich
- Dazu `Can-Retransform-Classes` im Manifest auf `true` setzen

Retransformation

- Retransformation auch möglich
- Dazu `Can-Retransform-Classes` im Manifest auf `true` setzen
- Transformer registrieren: `Instrumentation.addTransformer(ClassFileTransformer transformer)`

Retransformation

- Retransformation auch möglich
- Dazu `Can-Retransform-Classes` im Manifest auf `true` setzen
- Transformer registrieren: `Instrumentation.addTransformer(ClassFileTransformer transformer)`
- Und entsprechende Methode aufrufen: `Instrumentation.retransformClasses(Class<?>... classes)`

Retransformation

- Retransformation auch möglich
- Dazu `Can-Retransform-Classes` im Manifest auf `true` setzen
- Transformer registrieren: `Instrumentation.addTransformer(ClassFileTransformer transformer)`
- Und entsprechende Methode aufrufen: `Instrumentation.retransformClasses(Class<?>... classes)`
- Ja, so einfach ist es wirklich ;-)

Beispiel „Alle JUnit-Tests bestehen“

```
public class ClassToTest {

  public String getTheCanonicalClassName() {
    return "Falscher Name";
    //return this.getClass().getCanonicalName();
  }

  public int add(int a, int b) {
    return a * b;
    //return a + b;
  }
}
```

Die JUnit-Tests

```
public class JunitTests {

  @Test
  public void testClassName() {
    ClassToTest ctt = new ClassToTest();
    Assert.assertEquals("Falscher Klassenname",
                        ClassToTest.class.getCanonicalName(),
                        ctt.getTheCanonicalClassName());
  }

  @Test
  public void testAdd() {
    ClassToTest ctt = new ClassToTest();
    Assert.assertEquals("Falsche Summe", (3 + 4),
                                         ctt.add(3, 4));
  }
}
```

Der Transformer

```
public class JunitTransformer implements ClassFileTransforme

  @Override
  public byte[] transform(ClassLoader loader, String classNa
  Class<?> classBeingRedefined, ProtectionDomain protectionD
  byte[] classfileBuffer) throws IllegalClassFormatExceptio
    if (className.equals("org/junit/Assert")) {
      return transformAssert(); // ohne Exception-Handling
    }
    // alle anderen Klassen unveraendert zurueckgeben
    return classfileBuffer;
  }

  ...
}
```

Der Transformer

```
private byte[] transformAssert() throws Exception {
  ClassPool pool = ClassPool.getDefault();
  CtClass cc = pool.get("org.junit.Assert");
  for (CtMethod ctMethod : cc.getMethods()) {
      if (ctMethod.getName().startsWith("assert")) {
      ctMethod.setBody("return;");
    } else {
      // die anderen (equals(), clone(), wait(), ...)
    }
  }
  return cc.toBytecode();
}
```

Der Agent

```
public class TransformerAgent {

  public static void agentmain(String agentArgs, Instrumenta
    instrumentation.addTransformer(new JunitTransformer(),

    Class<?>[] classes = instrumentation.getAllLoadedClasses
    for (Class<?> c : classes) {
      if (c.getName().equals("org.junit.Assert")) {
        try {
          instrumentation.retransformClasses(c);
        } catch (UnmodifiableClassException e) {
          e.printStackTrace();
          System.err.println(c + " laesst sich nicht modifi
        }
      }
    }
  }
}
```

Und wie besteht man alle Unit-Tests?

```
public class ClassToTest {

  static {
    AgentLoader.loadAgent();
  }

  public String getTheCanonicalClassName() {
    return "Falscher Name";
    //return this.getClass().getCanonicalName();
  }

  public int add(int a, int b) {
    return a * b;
    //return a + b;
  }
}
```

Wozu ?

- Erstens: zum Spaß

Wozu ?

- Erstens: zum Spaß
- Zweitens: um in Gehaltsverhandlungen einzusteigen ;-)

Wozu ?

- Erstens: zum Spaß
- Zweitens: um in Gehaltsverhandlungen einzusteigen ;-)
- Drittens: um zu lernen

Wozu ?

- Erstens: zum Spaß
- Zweitens: um in Gehaltsverhandlungen einzusteigen ;-)
- Drittens: um zu lernen
- Viertens: . . .

Fragen und Anmerkungen