Class Transformer
One of the Best-Kept Java Secrets

Bernd Müller

Alternative Title

How to pass ALL unit tests ;-)

Speaker Introduction

- Prof. Computer Science, Ostfalia University, Germany
- Book author (JSF, JPA, JBoss Seam, ...)

- Member EGs JSR 344 (JSF 2.2) und JSR 338 (JPA 2.1)
- CEO PMST GmbH
- ...

Outline

- Class Loader
- Class Transformer
  - Redefinition
  - Retransformation

Motivation

- Bad News:
  - For application developers probably no utilizable knowledge
  - Framework developers will already know

Motivation

- Bad News:
  - For application developers probably no utilizable knowledge
  - Framework developers will already know
- Good News:
  - For Java developers it pays off to know what goes on behind the scenes
  - You will understand some things better and therefore get better
  - Eventually you will have some fun (see unit tests)

Class Loader

or

How classes get into the VM

Motivation: A Look Behind the Scenes

- Class loader working in background thus negligible for *Hello World* class programs

Motivation: A Look Behind the Scenes

- Class loader working in background thus negligible for *Hello World* class programs
- Shown for Java 1.1 FQN not sufficient: *Java is not type-safe*, Vijay Saraswat, AT&T Research, 1997

Motivation: A Look Behind the Scenes

- Class loader working in background thus negligible for *Hello World* class programs
- Shown for Java 1.1 FQN not sufficient: *Java is not type-safe*, Vijay Saraswat, AT&T Research, 1997
- In Java 1.2 new class loader architecture, remained valid since then

Motivation: A Look Behind the Scenes

- Class loader working in background thus negligible for *Hello World* class programs
- Shown for Java 1.1 FQN not sufficient: *Java is not type-safe*, Vijay Saraswat, AT&T Research, 1997
- In Java 1.2 new class loader architecture, remained valid since then
- Some methods in SDK classes have parameter of type `ClassLoader` and there are implementations of abstract class `ClassLoader`

Motivation: A Look Behind the Scenes

- Class loader working in background thus negligible for *Hello World* class programs
- Shown for Java 1.1 FQN not sufficient: *Java is not type-safe*, Vijay Saraswat, AT&T Research, 1997
- In Java 1.2 new class loader architecture, remained valid since then
- Some methods in SDK classes have parameter of type `ClassLoader` and there are implementations of abstract class `ClassLoader`
- Class loader important for Java EE. What the spec says ...

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

- VM
  - loads
  - links
  - and initializes

  classes and interfaces dynamically

Class Loader Basics

- VM
  - loads
  - links
  - and initializes

  classes and interfaces dynamically
- Loading: Find binary representation of class or interface type for some name and *create* class/interface out of this

Class Loader Basics

- VM
  - loads
  - links
  - and initializes

  classes and interfaces dynamically
- Loading: Find binary representation of class or interface type for some name and *create* class/interface out of this
- Linking: Insert class/interface in actual runtime state of VM to ensure it can be used

Class Loader Basics

- VM
  - loads
  - links
  - and initializes

  classes and interfaces dynamically
- Loading: Find binary representation of class or interface type for some name and *create* class/interface out of this
- Linking: Insert class/interface in actual runtime state of VM to ensure it can be used
- Initializing: Execute initialization method `<clinit>`

Loading and Linking

- Loading creates (with some basic checks) a `Class` object which can't be used yet

Loading and Linking

- Loading creates (with some basic checks) a `Class` object which can't be used yet
- Linking consists of:
  - Verification
  - Preparation
  - Resolution

Verification

- Verification checks that class will “behave well“ and does not cause runtime problems

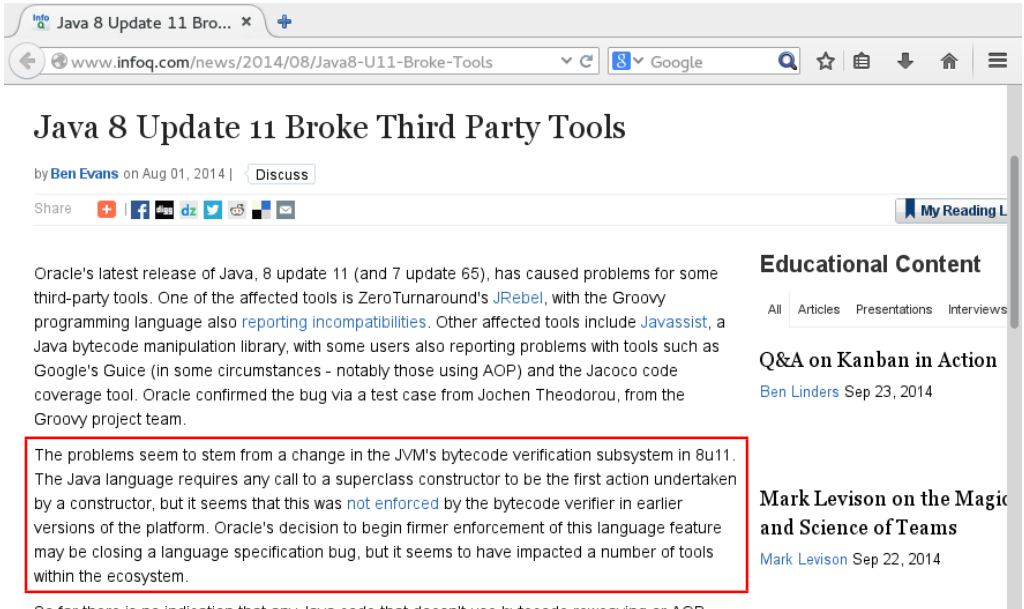
Verification

- Verification checks that class will “behave well“ and does not cause runtime problems
- Some checks:
  - Constant Pool consistent
  - No overriding of final methods
  - Methods respect access control
  - Methods called with correct number and types of parameters
  - Byte code does not manipulate stack
  - Variables initialized before usage
  - Values assigned to variables type compatible
  - Call to superclass constructor first statement in constructor
  - ...

Verification

- Verification checks that class will “behave well“ and does not cause runtime problems
- Some checks:
  - Constant Pool consistent
  - No overriding of final methods
  - Methods respect access control
  - Methods called with correct number and types of parameters
  - Byte code does not manipulate stack
  - Variables initialized before usage
  - Values assigned to variables type compatible
  - Call to superclass constructor first statement in constructor
  - ...
- If violated a `VerifyError` is thrown

Verfication: Recent Example



Preparation und Resolution

- Preparation:
  Ensures storage is allocated for class and class variables can be initialized.
  However, initialization is not performed and no byte code is executed

Preparation und Resolution

- Preparation:
  Ensures storage is allocated for class and class variables can be initialized.
  However, initialization is not performed and no byte code is executed
- Resolution:
  Checks that all referenced classes are loaded. If not these classes get loaded

Preparation und Resolution

- Preparation:
  Ensures storage is allocated for class and class variables can be initialized. However, initialization is not performed and no byte code is executed
- Resolution:
  Checks that all referenced classes are loaded. If not these classes get loaded
- If all classes are loaded, initialization is performed (class variables and static initializer blocks)

Preparation und Resolution

- Preparation:
  Ensures storage is allocated for class and class variables can be initialized. However, initialization is not performed and no byte code is executed
- Resolution:
  Checks that all referenced classes are loaded. If not these classes get loaded
- If all classes are loaded, initialization is performed (class variables and static initializer blocks)
- If completed class is ready to be used

Java 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.

Java 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 (cont'd)

- Chicken and egg problem: creation of class loader instance requires that class loader is already loaded. Who did it?

Class loader basics (cont'd)

- Chicken and egg problem: creation of class loader instance requires that class loader is already loaded. Who did it?
- Class Loader is ordinary class and extends `Object`. `Object` must already be loaded. Who did it?

Class loader basics (cont'd)

- Chicken and egg problem: creation of class loader instance requires that class loader is already loaded. Who did it?
- Class Loader is ordinary class and extends `Object`. `Object` must already be loaded. Who did it?
- Mentioned in Java doc: *bootstrap class loader*

Class loader basics (cont'd)

- Chicken and egg problem: creation of class loader instance requires that class loader is already loaded. Who did it?
- Class Loader is ordinary class and extends `Object`. `Object` must already be loaded. Who did it?
- Mentioned in Java doc: *bootstrap class loader*
- Because of delegation model a hierarchy is formed: All class loaders have a parent. Exception is bootstrap class loader

Class Loader Hierarchy

- **Bootstrap class loader**
  - Very early instantiated at VM start
  - Usually implemented native
  - Virtually belongs to VM
  - Loads system JARs, e.g. `rt.jar`
  - Does not verify
  - Path property: `sun.boot.class.path`

Class Loader Hierarchy

- **Bootstrap class loader**
  - Very early instantiated at VM start
  - Usually implemented native
  - Virtually belongs to VM
  - Loads system JARs, e.g. `rt.jar`
  - Does not verify
  - Path property: `sun.boot.class.path`
- **Extension class loader**
  - Loads standard extensions
  - Path property: `java.ext.dirs`
  - Class: `sun.misc.Launcher$ExtClassLoader`

Class Loader Hierarchie (cont'd)

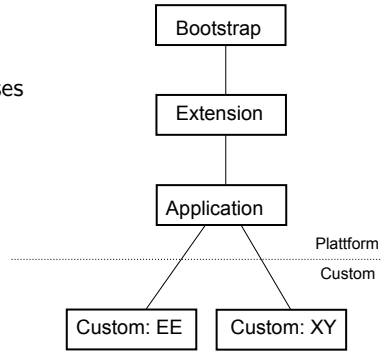
- **Application class loader**
  - Loads application classes
  - In SE class loader which loads most of the classes
  - Path property: `java.class.path`
  - Class: `sun.misc.Launcher$AppClassLoader`

Class Loader Hierarchie (cont'd)

- **Application class loader**
  - Loads application classes
  - In SE class loader which loads most of the classes
  - Path property: `java.class.path`
  - Class: `sun.misc.Launcher$AppClassLoader`
- **Custom class loader**
  - Required in Java-EE to comply with Spec
  - Everyone (you ?) can write own class loader
  - Well known example: JBoss Modules

Class Loader Hierarchie (cont'd)

- **Application class loader**
  - Loads application classes
  - In SE class loader which loads most of the classes
  - Path property: `java.class.path`
  - Class: `sun.misc.Launcher$AppClassLoader`
- **Custom class loader**
  - Required in Java-EE to comply with Spec
  - Everyone (you ?) can write own class loader
  - Well known example: JBoss Modules



Can we review this?

- VM class loader option: -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/.../jre/lib/rt.jar]
223: [Loaded java.lang.ClassLoader$3 from /usr/lib/jvm/.../jre/lib/rt.jar]
228: [Loaded java.lang.ClassLoader$NativeLibrary from /usr/lib/jvm/.../jre/lib/rt.jar]
243: [Loaded java.security.SecureClassLoader from /usr/lib/jvm/.../jre/lib/rt.jar]
244: [Loaded java.net.URLClassLoader from /usr/lib/jvm/.../jre/lib/rt.jar]
245: [Loaded sun.misc.Launcher$ExtClassLoader from /usr/lib/jvm/.../jre/lib/rt.jar]
247: [Loaded java.lang.ClassLoader$ParallelLoaders from /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.
256: [Loaded java.net.URLClassLoader$7 from /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.3.0.fc19.x86
259: [Loaded sun.misc.Launcher$ExtClassLoader$1 from /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.3.0
317: [Loaded sun.misc.Launcher$AppClassLoader from /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.3.0.f
318: [Loaded sun.misc.Launcher$AppClassLoader$1 from /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.3.0
319: [Loaded java.lang.SystemClassLoaderAction from /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.3.0.
321: [Loaded java.net.URLClassLoader$1 from /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.3.0.fc19.x86
383: [Loaded de.pdbm.simple.Main from file:/home/bernd/lehre/skripte/classloader/workspace/class-loac
384: [Loaded java.lang.Void from /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.3.0.fc19.x86_64/jre/lib
385: Hello World
```

Class Loader Delegation

- `ClassLoader` is super class of all class loaders

Class Loader Delegation

- `ClassLoader` is super class of all class loaders
- `ClassLoader` contains: `private final ClassLoader parent;`

Class Loader Delegation

- `ClassLoader` is super class of all class loaders
- `ClassLoader` contains: `private final ClassLoader parent;`
- Quite nice: `parent` comment (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` is super class of all class loaders
- `ClassLoader` contains: `private final ClassLoader parent;`
- Quite nice: `parent` comment (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.
```

- And Constructor for delegation to parent:
  `protected ClassLoader(ClassLoader parent)`

Class Loader Delegation

- `ClassLoader` is super class of all class loaders
- `ClassLoader` contains: `private final ClassLoader parent;`
- Quite nice: `parent` comment (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.
```

- And Constructor for delegation to parent:

```
protected ClassLoader(ClassLoader parent)
```

- And method

```
protected Class<?> loadClass(String name,
                             boolean resolve)
```

Class Loader Delegation

- `ClassLoader` is super class of all class loaders
- `ClassLoader` contains: `private final ClassLoader parent;`
- Quite nice: `parent` comment (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.
```

- And Constructor for delegation to parent:

```
protected ClassLoader(ClassLoader parent)
```

- And method

```
protected Class<?> loadClass(String name,
                             boolean resolve)
```

- If parent `null`: bootstrap class loader

Java Doc Method `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.

Java Doc Method `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.

- Called *parent first* strategy

Delegation to 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());     // AppClassLoader
System.out.println(loader2.getParent());     // AppClassLoader
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()); // AppClassLoader
System.out.println(class2.getClassLoader()); // AppClassLoader
```

Pre-Main

or

Is there Life before `main()` ?

Package `java.lang.instrument`

- Package `java.lang.instrument` was introduced with Java 5

Package `java.lang.instrument`

- Package `java.lang.instrument` was introduced with Java 5
- From Java 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.*“

Package `java.lang.instrument`

- Package `java.lang.instrument` was introduced with Java 5
- From Java 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. . . .* “

Package `java.lang.instrument`

- Package `java.lang.instrument` was introduced with Java 5
- From Java 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. . . .* “
- Intended for monitoring (as per Wikipedia)

Package `java.lang.instrument`

- Package `java.lang.instrument` was introduced with Java 5
- From Java 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. . . .* “
- Intended for monitoring (as per Wikipedia)
- Can be used for arbitrary tasks (e.g. by JPA provider, test coverage tools, . . .)

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.*“

- Central: the agent

Agent

- Deployed as JAR file

Agent

- Deployed as JAR file
- Manifest attribute defines agent class

Agent

- Deployed as JAR file
- Manifest attribute defines agent class
- Alternatives to start agent

Agent

- Deployed as JAR file
- Manifest attribute defines agent class
- Alternatives to start agent
  - Command line at VM start (required for command line implementations)

Agent

- Deployed as JAR file
- Manifest attribute defines agent class
- Alternatives to start agent
  - Command line at VM start (required for command line implementations)
  - After VM start by some not specified binding (optional and implementation dependent)

Start Agent via Command Line

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

Start Agent via Command Line

- Syntax: `-javaagent:`*jarpath*`[=`*options*`]`
- Allowed multiple times ⇒ multiple agents

Start Agent via Command Line

- Syntax: `-javaagent:`*jarpath*`[=`*options*`]`
- Allowed multiple times ⇒ multiple agents
- Manifest contains attribute `Premain-Class`

Start Agent via Command Line

- Syntax: `-javaagent:`*jarpath*`[=`*options*`]`
- Allowed multiple times ⇒ multiple agents
- Manifest contains attribute `Premain-Class`
- Agent class contains `premain()` method

Start Agent via Command Line

- Syntax: `-javaagent:`*jarpath*`[=`*options*`]`
- Allowed multiple times ⇒ multiple agents
- Manifest contains attribute `Premain-Class`
- Agent class contains `premain()` method
- After VM is initialized all `premain()` methods are executed in sequence, then `main()` method

Start Agent via Command Line

- Syntax: `-javaagent:`*jarpath*`[=`*options*`]`
- Allowed multiple times ⇒ multiple agents
- Manifest contains attribute `Premain-Class`
- Agent class contains `premain()` method
- After VM is initialized all `premain()` methods are executed in sequence, then `main()` method
- Two signatures allowed:

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

- Second called only if first doesn't exist

Agent start via command line (cont'd)

- Optional `agentmain()` method to use after VM start

Agent start via command line (cont'd)

- Optional `agentmain()` method to use after VM start
- If start via command line `agentmain()` is not called

Agent start via command line (cont'd)

- Optional `agentmain()` method to use after VM start
- If start via command line `agentmain()` is not called
- Agent loaded by system class loader

Agent start via command line (cont'd)

- Optional `agentmain()` method to use after VM start
- If start via command line `agentmain()` is not called
- Agent loaded by system class loader
- Each agent get's his parameters by `agentArgs` parameter. *One* String, e.g. agent has to parse himself

Agent start via command line (cont'd)

- Optional `agentmain()` method to use after VM start
- If start via command line `agentmain()` is not called
- Agent loaded by system class loader
- Each agent get's his parameters by `agentArgs` parameter. *One* String, e.g. agent has to parse himself
- If agent can not be loaded or `premain()` method does not exist VM is stopped

Agent start via command line (cont'd)

- Optional `agentmain()` method to use after VM start
- If start via command line `agentmain()` is not called
- Agent loaded by system class loader
- Each agent get's his parameters by `agentArgs` parameter. *One* String, e.g. agent has to parse himself
- If agent can not be loaded or `premain()` method does not exist VM is stopped
- Exceptions in `premain()` method also terminates VM

What can we do with that?

- Already mentioned: instrument something

What can we do with that?

- Already mentioned: instrument something
- E.g. to

What can we do with that?

- Already mentioned: instrument something
- E.g. to
  - Monitor

What can we do with that?

- Already mentioned: instrument something
- E.g. to
  - Monitor
  - Build proxies (JPA: associations, automatic dirty checking,...)

What can we do with that?

- Already mentioned: instrument something
- E.g. to
  - Monitor
  - Build proxies (JPA: associations, automatic dirty checking,...)
  - Delete `final` modifier

What can we do with that?

- Already mentioned: instrument something
- E.g. to
  - Monitor
  - Build proxies (JPA: associations, automatic dirty checking,...)
  - Delete `final` modifier
  - Mark paths to compute code coverage in unit tests

What can we do with that?

- Already mentioned: instrument something
- E.g. to
  - Monitor
  - Build proxies (JPA: associations, automatic dirty checking,...)
  - Delete `final` modifier
  - Mark paths to compute code coverage in unit tests
  - ...

What can we do with that?

- Already mentioned: instrument something
- E.g. to
  - Monitor
  - Build proxies (JPA: associations, automatic dirty checking,...)
  - Delete `final` modifier
  - Mark paths to compute code coverage in unit tests
  - ...
  - In general: attach desirable behavior afterwards and only if needed

Monitoring Example: Number of Method Invocations

```
public class ClassToMonitor {

  public void foo() {
    // something
  }

}
```

- Task: Count number of `foo()` invocations

Monitoring Example: Counter and Main

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

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

Monitoring Example: the Agent

```
public class MonitorAgent {

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

In MANIFEST.MF:

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

Monitoring Example: Instrumentation with Javassist

```
public class MonitorTransformer
             implements ClassFileTransformer {

  public byte[] transform(ClassLoader loader, String className,
      Class<?> classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] classfileBuffer) throws IllegalClassFormatException {

    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 | IOException e) {
        ...
      }
    }
    return classfileBuffer; // other classes unchanged
  }
}
```

Attach API

or

How to talk with a VM ?

Recap from 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.“

- Attention: implementation dependent
- But: available in HotSpot, JRockit, IBM SDK, SAP SDK
- Interface by abstract class `VirtualMachine` in package `com.sun.tools.attach` contained in `tools.jar`

Java Doc class `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.* “

- Factory method `attach(<pid>)` to get attached instance
- Method `loadAgent(<agent>,<args>)` to load agent and to start agent (method `agentmain()`)

Agent with agentmain()

- Agent

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

- Manifest sets attribute `Agent-Class`
- Attribute `Can-Redefine-Classes` `true` if redefinition (new class definition)
- Attribute `Can-Retransform-Classes` `true` if retransformation (change existing byte code)

Example: Change Method and Reload

```
public class ClassToBeRedefined {

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

}
```

Example: Cange Method and Reload (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
    if (Agent.instrumentation == null) {
      throw new RuntimeException("Agent not started. Instrumentation not po
    }
    Agent.instrumentation.redefineClasses(definitions);
  }
}
```

Example: Change Method and Reload (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();
  }
```

Example: Change Method and Reload (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) {
    ...
  }
}
```

Seeing is believing . . .

Demo

Java Doc `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).

Java Doc `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.

Java Doc `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 also possible

Retransformation

- Retransformation also possible
- Set manifest attribute `Can-Retransform-Classes` to `true`

Retransformation

- Retransformation also possible
- Set manifest attribute `Can-Retransform-Classes` to `true`
- Register Transformer:
  `Instrumentation.addTransformer(ClassFileTransformer transformer)`

Retransformation

- Retransformation also possible
- Set manifest attribute `Can-Retransform-Classes` to `true`
- Register Transformer:
  `Instrumentation.addTransformer(ClassFileTransformer transformer)`
- Call appropriate method:
  `Instrumentation.retransformClasses(Class<?>... classes)`

Retransformation

- Retransformation also possible
- Set manifest attribute `Can-Retransform-Classes` to `true`
- Register Transformer:
  `Instrumentation.addTransformer(ClassFileTransformer transformer)`
- Call appropriate method:
  `Instrumentation.retransformClasses(Class<?>... classes)`
- Yes — it's that simple !

Example: „Pass all Unit Tests“

```
public class ClassToTest {

  public String getTheCanonicalClassName() {
    return "Wrong name";
    //return this.getClass().getCanonicalName();
  }

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

The Tests

```
public class JunitTests {
  @Test
  public void testClassName() {
    ClassToTest ctt = new ClassToTest();
    Assert.assertEquals("Wrong class name",
                        ClassToTest.class.getCanonicalName(),
                        ctt.getTheCanonicalClassName());
  }

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

The Transformer

```
public class JunitTransformer implements ClassFileTransformer {

  @Override
  public byte[] transform(ClassLoader loader, String className,
  Class<?> classBeingRedefined, ProtectionDomain protectionDomain,
  byte[] classfileBuffer) throws IllegalClassFormatException {
    if (className.equals("org/junit/Assert")) {
      return transformAssert(); // w/o Exception-Handling
    }
    // other classes unmodified
    return classfileBuffer;
  }

  ...
}
```

The Transformer (cont'd)

```
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 {
      // the rest (equals(), clone(), wait(), ...)
    }
  }
  return cc.toBytecode();
}
```

Agent

```
public class TransformerAgent {
  public static void agentmain(String agentArgs, Instrumentation instrumenta
    instrumentation.addTransformer(new JunitTransformer(), true);
    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 + " not modifiable");
        }
      }
    }
  }
}
```

And how to activate ?

```
public class ClassToTest {

  static {
    AgentLoader.loadAgent();
  }

  public String getTheCanonicalClassName() {
    return "Wrong name";
    //return this.getClass().getCanonicalName();
  }
  public int add(int a, int b) {
    return a * b;
    //return a + b;
  }
}
```

Seeing is believing . . .

Demo

What is all this for ?

- First: to have some fun

What is all this for ?

- First: to have some fun
- Second: to start salary negotiations ;-)

What is all this for ?

- First: to have some fun
- Second: to start salary negotiations ;-)
- Third: to learn

What is all this for ?

- First: to have some fun
- Second: to start salary negotiations ;-)
- Third: to learn
- Fourth: ...

Questions and Remarks