Evgeny Mandrikov

Follow me! @_Godin_

GitHub.com/Godin

one of JaCoCo and Eclipse EclEmma Project Leads
Disclaimer

/* TODO don't forget to add huge disclaimer that
All opinions hereinbelow are my own and not my employers.
They can only dream that they own them. */
Java Code Coverage Mechanics
Java
Code
Coverage
Mechanics
Java
Code Coverage
Mechanics
Java
Code
Coverage
Mechanics
Real Disclaimer

Blood and guts of JVM

Strong language - bytecode

Intense violence for brain

Implementation details!!!
EMMA

EclEmma

JaCoCo

- 2004
- 2005
- 2006
- 2007
- 2008
- 2009
- 2010
- 2011
- 2012
- 2013
- 2014
- 2015
- 2016
- 2017

- 5
- 6
- 7
- 8
- 9
Observer Effect

“

In physics, the term observer effect refers to changes that the act of observation will make on a phenomenon being observed. This is often the result of instruments that, by necessity, alter the state of what they measure in some manner.

” Wikipedia
Code Coverage

- runtime profiling
  - JVMPI
  - JVMTI

- instrumentation
  - source
  - bytecode
    - offline
    - on-the-fly
JaCoCo works on class files only

```
public static IRuntime createFor(final Instrumentation
final String className, final String accessField
throws ClassNotFoundException {
final ClassFileTransformer transformer = new ClassFileTransformer
public byte[] transform(final ClassLoader loader
final String name, final Class<?> clazz
final ProtectionDomain protectionDomain
throws IllegalArgumentException {
if (name.equals(className)) {
return instrument(source, accessFieldName);
return null;
}
instrument(source, accessFieldName);
clazz = Class.forName(className, true, loader);
inst.addTransformer(transformer);
inst.removeTransformer(transformer);
try {
clazz.getField(accessFieldName);
} catch (final NoSuchFieldException e) {
throw new RuntimeException("Class \%s could not be instrumented.",
return new ModifiedSystemClassRuntime(clazz, access);
```

As easy as setting an arg for the JVM

java -javaagent:jacocoagent.jar[=options] Application

class PreMain {
    public static void premain(
        String options,
        Instrumentation instrumentation
    ) throws Exception {
        instrumentation.addTransformer(...);
    }
}
JaCoCo Validation Test Suite

```java
// 6. Executed while block
int i = 0;
while (i++ < 3) { // $line-whiletruefalse$
    nop(); // $line-executedwhile$
}

// 7. Executed do while block
do {
    nop(); // $line-executeddowhile$
} while (f());

// 8. Missed for block
for (nop(); f(); nop()) { // $line-missedforincrementer$
    nop(); // $line-missedfor$
}

// 9. Executed for block
for (int j = 0; j < 1; j++) { // $line-executedforincrementer$
    nop(); // $line-executedfor$
}

// 10. Missed for each block
for (Object o : Collections.emptyList()) { // $line-missedforeachincrementer$
    nop(o); // $line-missedforeach$
}
```
Tested on 5 major JDK versions

jacoco / jacoco 🟢 build passing

- Do not violate JVMS regarding initialization of final fields
  
  Without this change instrumented classes can't pass checks and cause IllegalAccessError starting from OpenJDK 9 EA b127 (see https://bugs.openjdk.java.net/browse/JDK-8157181).
  
  ⬤ Commit 06f52f0
  ⬤ Compare ba923a0..06f52f0
  ⬤ Evgeny Mandrikov authored and committed

Build Jobs

- # 1142.1  
  ⤧ JDK=5

- # 1142.2  
  ⤧ JDK=6

- # 1142.3  
  ⤧ JDK=7

- # 1142.4  
  ⤧ JDK=8

- # 1142.7  
  ⤧ JDK=9-ea-stable
Not only issues in JaCoCo...

<table>
<thead>
<tr>
<th>T</th>
<th>Key</th>
<th>Summary</th>
<th>Assignee</th>
<th>Reporter</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JDK-8154017</td>
<td>Shutdown hooks are racing against shutdown sequence, if System.exit()-calling thread is interrupted</td>
<td>Chris Hegarty</td>
<td>Aleksey Shipilev</td>
<td>CLOSED</td>
</tr>
<tr>
<td></td>
<td>JDK-8164302</td>
<td>No initialization for super interface with default method</td>
<td>Karen Kinnear</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JDK-8080555</td>
<td>Different bytecode between JDK8u45 and JDK8u60-ea-b12</td>
<td>Vicente Arturo Romanero Zaldivar</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JDK-8134862</td>
<td>JVM crash at PhaselIdealLoop::idom_no_update</td>
<td>Balchandra Vaidya</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JDK-8073658</td>
<td>Invalid annotations in bridge methods</td>
<td>Srikanth Adayapalam</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JDK-8131041</td>
<td>Garbage in output of DecimalFormat</td>
<td>Naoto Sato</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JDK-8163969</td>
<td>Cyclic interface initialization causes JVM crash</td>
<td>Coleen Phillimore</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Details**

- **Type:** Bug
- **Status:** CLOSED
- **Priority:** P2
- **Resolution:** Fixed
- **Affects Version/s:** 7, 8, 9
- **Fix Version/s:** 9
- **Component/s:** core-libs
- **Labels:** autoverify, foss-libs, jacoco-found
- **Subcomponent:** java.lang
- **Resolved In Build:** b125
- **Verification:** Verified
Probes
ALWAYS WONDERED... JUST WHICH OF US... WAS FASTEST...
Probe

- Minimal runtime overhead
- No side effects on application code
- Thread safe
- Record execution
- Identification

```java
probes[id] = true;
ALOADx probes
xPUSH id
ICONST_1
BASTORE
```
Bytecode of Assertions

class Fun {
    void fun(boolean x) {
        assert x;
    }
}

// javap -c Fun
    getstatic $assertionsDisabled
    ifne L
    iload_1
    ifne L
    new java/lang/AssertionError
    dup
    invokespecial <init>()V
    athrow
L:
    return
Bytecode of Assertions

class Fun {
    static final synthetic boolean $assertionsDisabled = Fun.class.desiredAssertionStatus();

    void fun(boolean x) {
        if (! $assertionsDisabled)
            if (! x)
                throw new AssertionError();
    }
}
class Fun {
    static final synthetic boolean[] $jacocoData = ...;

    void fun() {
        boolean[] probes = $jacocoData;
        probes[0] = true;
        ...
        probes[...] = true;
    }
}

Nice idea!
class Outer {
    private int counter;

    class Inner {
        void inc() {
            counter++;
        }
    }
}

// javap -v Outer
synthetic static private
int access$008(Outer o) {
    return o.counter++;
}

// javap -c Outer$Inner
synthetic final Outer this$0;

void inc() {
    Outer.access$008(this$0)
Methods in Enums

defined

```java
enum Fun { 
    C
}

// javap -v Fun

synthetic static
Fun valueOf(java.lang.String);

synthetic static
Fun[] values();
```
class Fun {
    void exec(Runnable task) {
        task.run();
    }
    void fun() {
        exec(() -> {
            ...
        });
    }
}

// javap -c -p -l Fun
private static synthetic lambda$fun$0() {
    ...
}

new ClassReader(classBytes)

return \texttt{downgrade}(\texttt{classBytes})
? \texttt{upgrade}(\texttt{transform}(\texttt{classBytes}))
: \texttt{transform}(\texttt{classBytes});
Enjoy Hardcore
class Child extends Base {
    void someMethod() {
        assert 1 == 2;
    }
}

class Base {
    static {
        new Child().someMethod();
    }
}

public static void main(String[] args) {
    new Child();
}
Bad Cycle (problem)

class Fun {
    static final synthetic boolean[] $jacocoData = ... ;

    void fun() {
        boolean[] probes = $jacocoData;
        probes[0] = true;  // NullPointerException
        ...
        probes[...] = true;
    }
}
class Fun {
    static final synthetic boolean[] $jacocoData;

    static synthetic boolean[] $jacocoInit() {
        if ($jacocoData == null) $jacocoData = ... ;
        return $jacocoData;
    }

    void fun() {
        boolean[] probes = $jacocoInit();
        ...
    }
}
6.5. putstatic

if the field is final, it must be declared in the current class, and the instruction must occur in the <clinit> method of the current class. Otherwise, an IllegalAccessException is thrown

Checked in JDK 9 EA b127 for class files version 53.
Bad Cycle (correct solution)

class Fun {
    static final synthetic boolean[][] $jacocoData;

    static synthetic boolean[][] $jacocoInit() {
        if ($jacocoData == null) $jacocoData = ...;
        return $jacocoData;
    }

    void fun() {
        boolean[][] probes = $jacocoInit();
        ...
    }
}
Interfaces

interface Fun {
    static final
    Object field;

    static {
        field = ... ;
    }
}

interface Fun {
    default method() {
        // oups???
    }
}
"12.4.1. When Initialization Occurs

A class or interface type T will be initialized immediately before the first occurrence of any one of the following:

... an instance of T is created...
... static method declared by T is invoked...

When a class is initialized, its superclasses are initialized, as well as any superinterfaces that declare any default methods. Initialization of an interface does not, of itself, cause initialization of any of its superinterfaces."
interface Base {
    Object o = new Child() {
        .someMethod();
    }
    default void base() {
        // JLS 12.4.1
    }
}

default Object someMethod() {
    throw ...;
}
Bad Cycle with Interfaces (problem)

- base clinit → child method → child clinit
- < JDK 8u40 <=

class: child clinit → base clinit → child method
  + crash because of exception

static: base clinit → child method → child clinit

- < JDK 8u152 EA <=
  child clinit → base clinit → child method
Bad Cycle with Interfaces (solution)

interface Fun {
    static final synthetic boolean[] $jacocoData = ... ;

    static synthetic boolean[] $jacocoInit() {
        return $jacocoData == null
            ? ... // slow path without assignment for JDK < 8u152
            : $jacocoData ;
    }

    default void fun() {
        boolean[] probes = $jacocoInit();
        ...
    }
}
Class Identification

Class c1 = ...;
Class c2 = ...;

( c1.getName() ).equals( c2.getName )
⇔ is c1 the same as c2 ???

long classId = CRC64(classBytes);
class RestrictiveClassLoader extends ClassLoader {
    protected Class<?> loadClass(String name, boolean resolve) throws ClassNotFoundException {
        if (!name.startsWith("java/") && !name.startsWith("org/sonarsource/"))
            throw new ClassNotFoundException();
        return super.loadClass(name, resolve);
    }
}

Runtime Access
boolean[] probes = ??? ;

// Oups
// can use only classes
// “java/**”
Runtime Access (Solution)

Object access =
    java.util.UUID.$jacocoAccess; // created at agent startup

Object[] args = new Object[] {
    classId,     // Long
    className,   // String
    probesCount  // Integer
};

access.equals(args);

boolean[] probes = (boolean[]) args[0];
Get Involved and Have Fun

- https://jdk9.java.net/download/
- https://groups.google.com/forum/#!forum/jacoco
- StackOverflow
- https://github.com/eclipse/eclemma
- https://github.com/jacoco/jacoco
KEEP CALM
and keep clapping
THIS IS THE END
SERVER CRASHED?

PLEASE TELL ME YOU DON'T HAVE A BACKUP
Principles and Best Practices

- Don’t care about percentage value, observe the amount of untested code
- Focus on coverage of modified and new code
- Make coverage analysis an inherent part of your build/test chain
- Make coverage reports available to everybody in the team
  - But not the management!
- Always go for functional coverage when writing tests
  - Just executing code will not improve its quality!
Common Pitfalls and how to avoid them

- Different Class Files runtime/analysis (recall class ids)
  - different compiler implementations
  - different compiler versions
  - different compiler settings
  - Pack200
- Different version of the same class in same group
- Not graceful JVM termination
- Reflection (recall synthetic fields and methods)
- Pre-instrumentation requires direct dependency on the JaCoCo runtime
- Interoperability with PowerMock
  - overrides JaCoCo init method
  - might bypass agents, because reads class files as resources