

at **DEVOXX™ France**

by Evgeny Mandrikov

# Java Code Coverage Mechanics



# Evgeny Mandrikov



Followme! @\_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**



**1.x**



**JaCoCo**



**3.x**

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

☕ 5

☕ 6

☕ 7

☕ 8

☕ 9

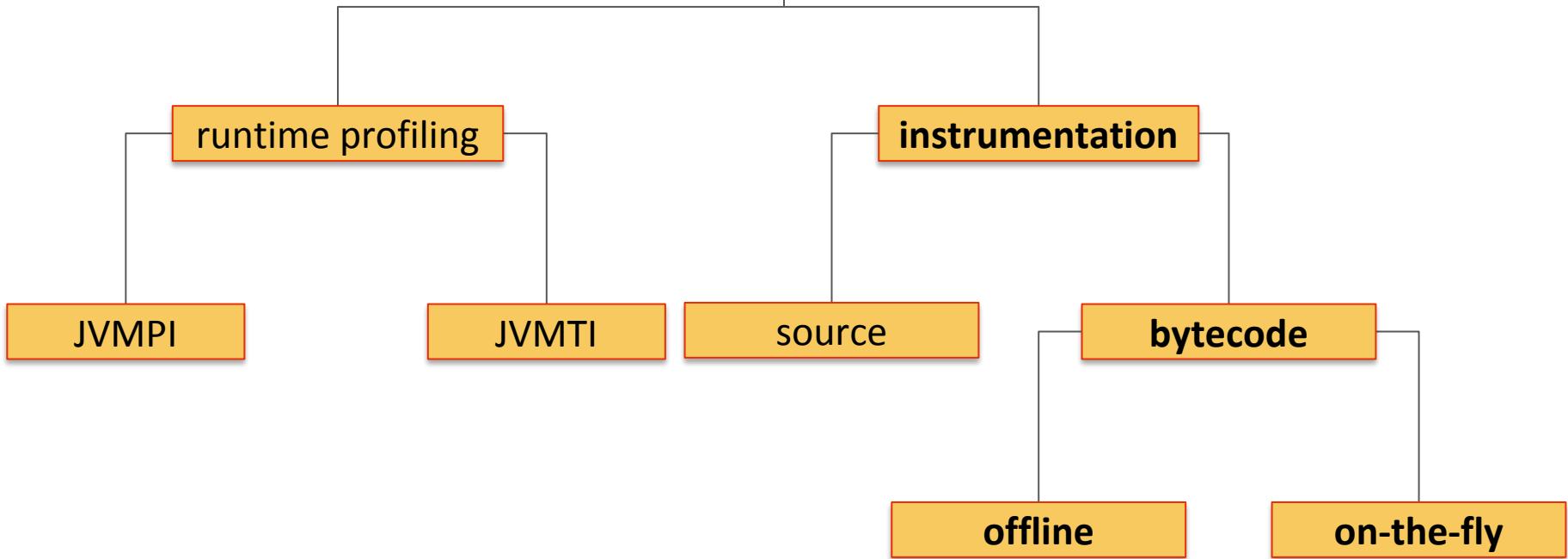
# Observer Effect

"

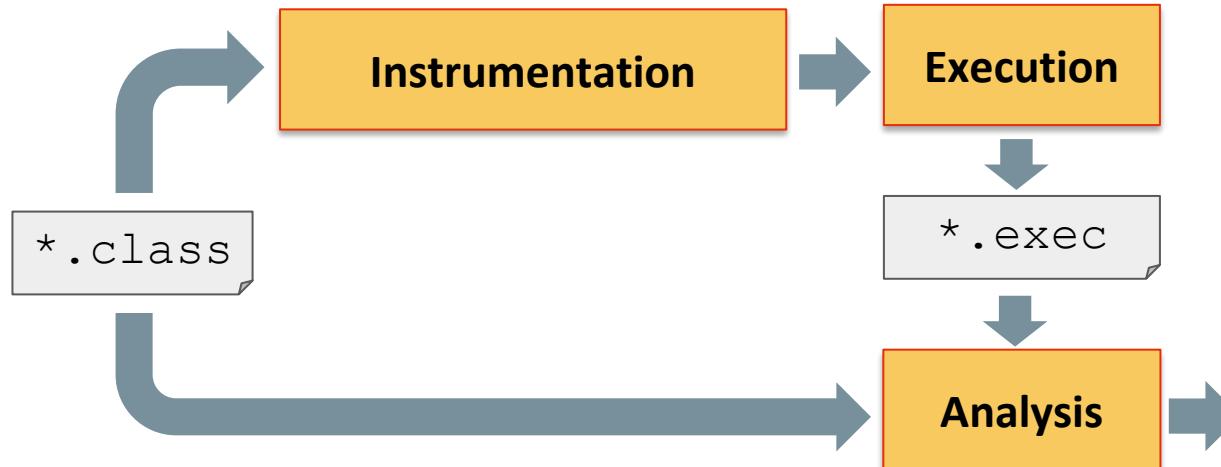
In physics, the term observer effect refers to changes that the act of observation will make on a phenomenon being observed. This is often the result of **instruments that**, by necessity, **alter the state of what they measure** in some manner.

" Wikipedia

# Code Coverage



JaCoCo works on class files only



```
120.     public static IRuntime createFor(final Instrumentation
121.             final String className, final String accessFieldName)
122.             throws ClassNotFoundException {
123.                 final ClassFileTransformer transformer = new ClassFileTransformer() {
124.                     public byte[] transform(final ClassLoader loader,
125.                         final String name, final Class<?> clazz,
126.                         final ProtectionDomain protectionDomain
127.                         throws IllegalClassFormatException {
128.                             if (name.equals(className)) {
129.                                 return instrument(source, accessFieldName);
130.                             }
131.                         return null;
132.                     }
133.                 };
134.                 inst.addTransformer(transformer);
135.                 final Class<?> clazz = Class.forName(className.replace(
136.                     inst.removeTransformer(transformer);
137.                     try {
138.                         clazz.getField(accessFieldName);
139.                     } catch (final NoSuchFieldException e) {
140.                         throw new RuntimeException(format(
141.                             "Class %s could not be instrumented.", clazz.getName());
142.                     }
143.                 }
144.             return new ModifiedSystemClassRuntime(clazz, accessFieldName);
145.         }
```

# 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

```
// 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 

Current Branches Build History Pull Requests > Build #1064

- ✓ Do not violate JVMS regarding initialization of final fields

Without this change instrumented classes can't pass checks  
and cause `IllegalAccessException` starting from OpenJDK 9 EA b127  
(see <https://bugs.openjdk.java.net/browse/JDK-8157181>).

 Commit 06f52f0

 Compare ba923a0..06f52f0

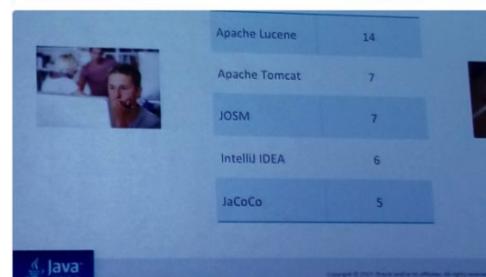
 Evgeny Mandrikov authored and committed

## Build Jobs

 # 1142.1	 JDK=5
 # 1142.2	 JDK=6
 # 1142.3	 JDK=7
 # 1142.4	 JDK=8
 # 1142.7	 JDK=9-ea-stable

Alexey Fyodorov  
@23derevo

Top external #OpenJDK issues submitters in past 6 months. #FOSDEM



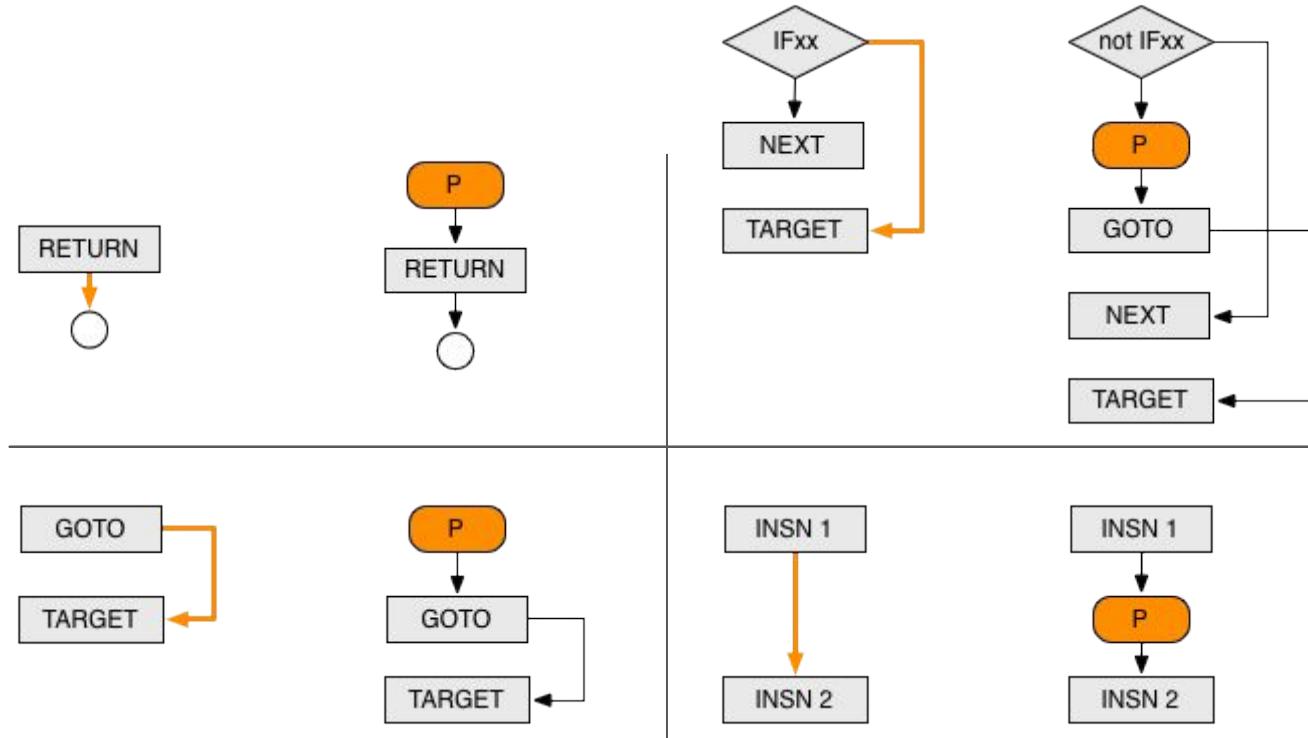
# Not only issues in JaCoCo...

T	Key	Summary	Assignee	Reporter	P ↓	Status
<a href="#">●</a>	JDK-8154017	Shutdown hooks are racing against shutdown sequence, if System.exit()-calling thread is interrupted	Chris Hegarty	Aleksey Shipilev	<a href="#">②</a>	<a href="#">CLOSED</a>
<a href="#">●</a>	JDK-8164302	No initialization for super interface with default method	Karen Kinnear			
<a href="#">●</a>	JDK-8080555	Different bytecode between JDK8u45 and JDK8u60-ea-b12	Vicente Arturo Romero Zaldivar			
<a href="#">●</a>	JDK-8134862	JVM crash at PhaseldealLoop::idom_no_update	Balchandra Vaidya			
<a href="#">●</a>	JDK-8073658	Invalid annotations in bridge methods	Srikanth Adayapalam			
<a href="#">●</a>	JDK-8131041	Garbage in output of DecimalFormat	Naoto Sato			
<a href="#">●</a>	JDK-8163969	Cyclic interface initialization causes JVM crash	Coleen Phillimore			

## Details

Type:	<a href="#">Bug</a>
Status:	<a href="#">CLOSED</a>
Priority:	<a href="#">② P2</a>
Resolution:	Fixed
Affects Version/s:	7, 8, 9
Fix Version/s:	9
Component/s:	core-libs
Labels:	<a href="#">autoverify</a> <a href="#">foss-libs</a> <a href="#">jacoco-found</a>
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

```
probes[id] = true;
```

```
ALOADx    probes  
xPUSH      id  
ICONST_1  
BASTORE
```

# Bytecode of Assertions

```
class Fun {                                // javap -c Fun
    void fun(boolean x) {
        assert x;
    }
}
```

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 AssertionException();  
    }  
}
```

# Nice idea!

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

# Bridge Methods

```
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

```
enum Fun {  
    C  
}  
  
// javap -v Fun  
  
synthetic static  
Fun valueOf(java.lang.String);  
  
synthetic static  
Fun[ ] values();
```

# Lambdas

```
class Fun {  
    void exec(Runnable task) {  
        task.run();  
    }  
    void fun() {  
        exec(() -> {  
            ...  
        });  
    }  
}
```

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

# ASM

```
// javac -version  
java version "9-ea"  
  
// javac Fun.java  
// javap -version  
1.7.0_80  
  
// javap -v Fun  
major version: 53  
  
// javap -c -XDdetails -cp bin:src Example
```

```
new ClassReader(classBytes)  
// IllegalArgumentException  
  
return downgrade(classBytes)  
? upgrade(transform(classBytes))  
: transform(classBytes);
```

Enjoy

*Hardcore*

# Bad Cycle

```
class Child extends Base {  
    void someMethod() {  
        assert 1 == 2;  
    }  
}
```

```
public static void main(String[] args) {  
    new Child();  
}
```

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

# Bad Cycle (problem)

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

# Bad Cycle (solution)

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

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

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

# Java Virtual Machine Specification

## “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 IllegalAccessError 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???  
    }  
}
```



# Java Language Specification

## “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.”

# Bad Cycle with Interfaces

```
interface Base {  
    Object o = new Child(){}  
        .someMethod();  
  
    default void base() {  
        // JLS 12.4.1  
    }  
}  
  
public static void main(String[] args) {  
    new Child(){}; // or Child.staticMethod();  
}
```

```
interface Child  
    extends Base {  
    Object o = new Object() {  
        { println("clinit"); }  
    };  
    default Object someMethod() {  
        throw ...;  
    }  
}
```

# Bad Cycle with Interfaces (problem)

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

class: child clinit → base clinit → child method

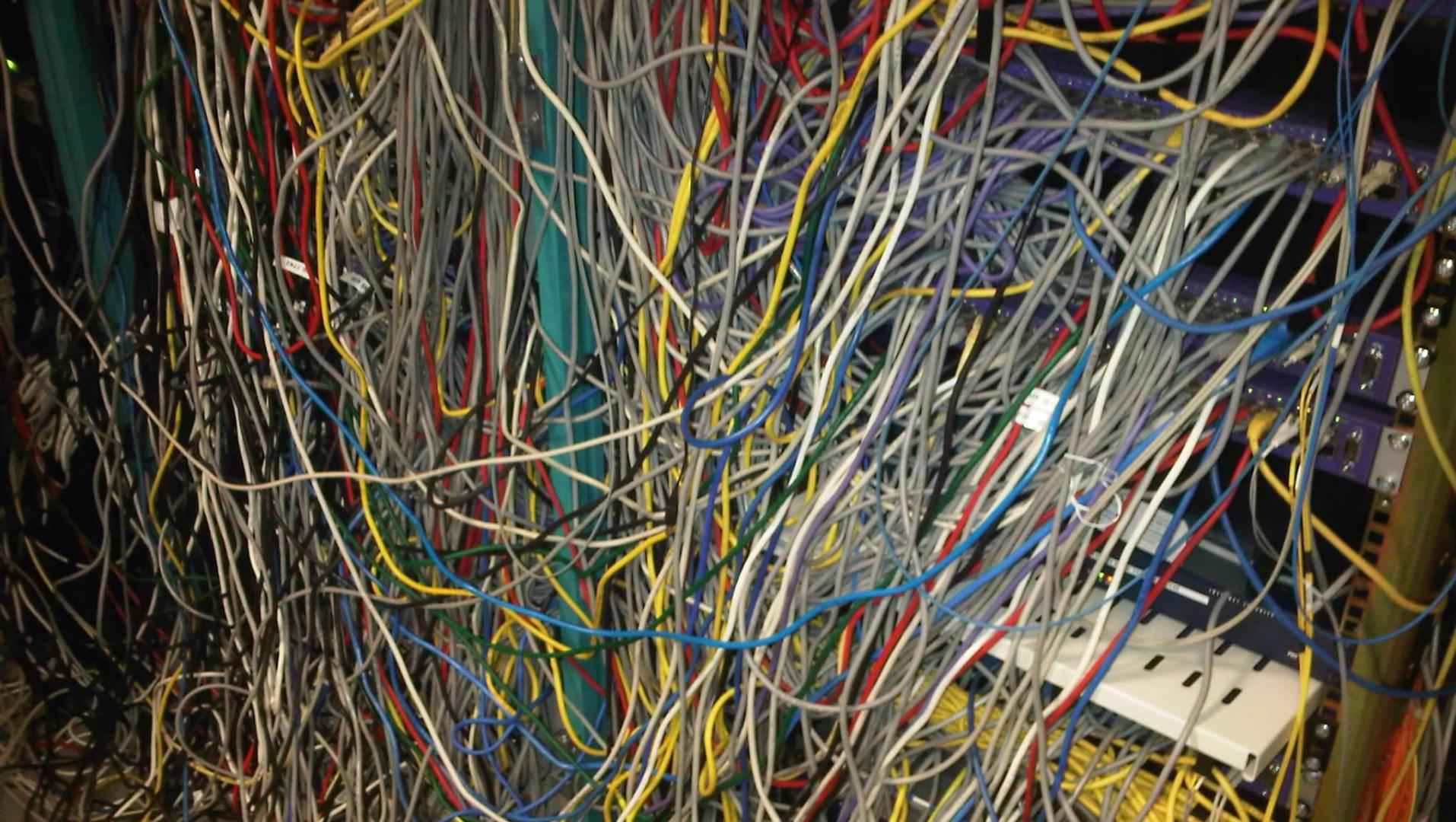
+ **crash** because of exception

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 ???

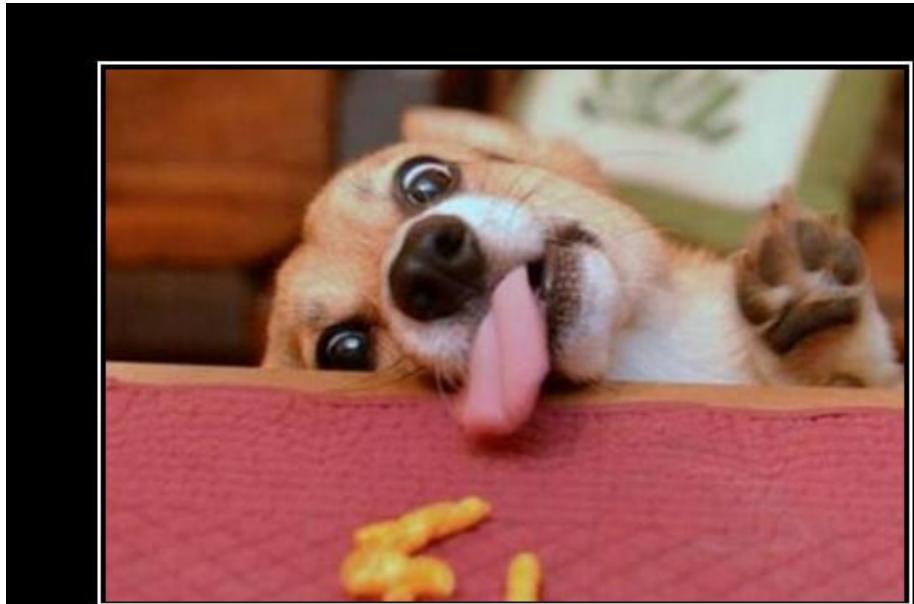
```
long classId = CRC64(classBytes);
```

# Runtime Access

```
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 (Problem)

```
boolean[ ] probes = ??? ;  
// Oups  
// can use only classes  
// "java/**"
```



S O C L O S E  
Yet too far.

# 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

