JavaOne™

ORACLE®

Java Code Coverage with JCov

Implementation Details and Use Cases

Alexandre (Shura) Iline
JDK Test Architect
JDK SQE team
Oct 1, 2014

JavaOne
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

JCov development

Alexey Fedorchenko

Dmitry Fazunenko

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Program Agenda

1. History, facts
2. Getting the data with JCov
3. Applied to OpenJDK
4. Using the data
5. Links, more information

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Facts

- JCov is a **J**ava Code **Cov**erage tool.
- The code coverage tool for JCK
- The code coverage tool for Oracle JDK.
- Other products:
  - JavaFX
  - SceneBuilder
  - …

JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

History

- 1996: First prototype
- 1997: Integration with JDK 1.1
- 1998: Used “in production”
- JDK 1.2
- …
- JDK 7
- 2014: Open-source as part of OpenJDK codetools project.
- 2014: JCov 3.0 – a public “release” supporting JDK 8
- JDK 9 – in progress

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

More on JCov dev

1.5 engineer at a time, on average :)

Leonid Arbouzov, Alexander Petrov, Vladimir Generalov, Serguei Chukhontsev, Oleg Uliankin, Gregory Steuck, Pavel Ozhdikhin, Konstantin Bobrovsky, Robert Field, Alexander Kuzmin, Leonid Mesnik, Sergey Borodin, Andrey Titov, Dmitry Fazunenko, Alexey Fedorchenko

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Code coverage is

- An information on what **source code** is **exercised in execution**

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

most often ...

Code coverage is

- An information on what source code is exercised **in testing**

where ...

Testing is

- Activities to prove that the code **behaves as expected**

JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Other possible usages.

- Usage monitoring
  - Save data in a test environment
- Check coverage from generated logic
  - Fuzzing
  - Load testing
  - Regression test generation

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Program Agenda

1. History, facts
2. Getting the data with JCov
3. Applied to OpenJDK
4. Using the data
5. Links, more information

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Getting the data

- A simplest use case
- A pick or two under the hood
- More possibilities
  - Dynamic vs. static instrumentation
  - Grabber
  - Individual test coverage
  - Abstract coverage
  - Drop points
  - Direct coverage
  - Running big test suites

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Simplest use case

demo

- Compile java files as usual

- "Instrument" the byte code

```
java -jar jcov.jar Instr <application classes>
```

- Run the code

```
java -classpath ...:jcov_file_saver.jar ...
```

- Create a report

```
java -jar jcov.jar RepGen <jcov xml file>
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A pick under the hood

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

JCov bytecode insertion

demo

A pick under the hood

```
public int getX();
  Code:
     0: ldc            #31   // int 2
     2: invokestatic   #29   \
         // Method com/sun/tdk/jcov/runtime/Collect.hit:(I)V
     5: aload_0
     6: getfield       #2                  // Field x:I
     9: ireturn
```

Warning: the code is a subject to change!!!

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

JCov bytecode insertion

As if it was in Java

```
public int getX() {
  com.sun.tdk.jcov.runtime.Collect.hit(2);
  return x;
}
```

Warning: the code is a subject to change!!!

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Increasing the count

demo

One step deeper

```
public class Collect {
  ...
  private static long counts[];
  ...
  public static void hit(int slot) {
    counts[slot]++;
  }
  ...
}
```


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Saving the data

Finally …

```
Runtime.getRuntime().addShutdownHook(new Thread() {
    public void run() {
        ...
        Collect.disable();
        Collect.saveResults();
        Collect.enable();
        ...
    }
});
```


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Performance impact

- Instrumentation phase:
  - proportional to the application code size.
- Execution phase:
  - Counters increments - very low. Depends on the size of blocks.
  - Saving data – from significant to blocking.
    - There is a cure! :) More on next slides.
- Report generation phase.

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

XML coverage data

demo

```
<class name="Point" supername="java/lang/Object" ...>
  ...
  <meth name="getX" vmsig="()I" ...>
    <bl s="0" e="4">
      <methenter s="0" e="4" id="5" count="1"/>
      <exit s="4" e="4" opcode="ireturn"/>
    </bl>
  </meth>
</class>
```

Warning: the format is a subject to change!!!

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Back to the simplest use case

demo

- Compile java files as usual

- "Instrument" the byte code

```
java -jar jcov.jar Instr <application classes>
```

- Run the code

```
java -classpath ...:jcov_file_saver.jar ...
```

- Create a report

```
java -jar jcov.jar RepGen <jcov xml file>
```

Incomplete data!!!

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Code coverage template

- Describes all code there is to cover

- Created during instrumentation:

```
java -jar jcov.jar Instr -t template.xml <in> <out>
```

- Created explicitly:

```
java -jar jcov.jar TmplGen -t template.xml <in>
```

- A regular JCov XML file.
  - Merge
  - Filter
  - etc. etc.

JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Simplest use case *corrected* *demo*

- Compile java files as usual
- "Instrument" the byte code

```
java -jar jcov.jar Instr -t template.xml <directory or jar>
```

- Run the code

```
java -classpath ...:jcov_file_saver.jar ...
```

- Merge with the template

```
java -jar jcov.jar Merger -o merge.xml template.xml result.xml
```

- Create a report

```
java -jar jcov.jar RepGen result.xml
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Simplest use case *with filtering*

demo

- Compile java code
- "Instrument" the byte code

```
java -jar jcov.jar Instr -t template.xml \
  -i com.company.product \
   <application classes>
```

- Run the code, merge with the template
- Create a report

```
java -jar jcov.jar RepGen \
  -e com.company.product.test result.xml
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

More on getting the data

JavaOne
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Dynamic instrumentation

demo

- Compile java files as usual

- Run with an extra VM option

```
java -classpath ... -javaagent:jcov.jar ...
```

- `result.xml` generated.

- Merge and filter by the template

demo

```
java -jar jcov.jar Merger -o merge.xml \
  -t template.xml result.xml
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Dynamic vs. static instrumentation

	Dynamic	Static
Modification of the tested application	Not needed	Needed
Coverage collected for	All code(*)	Instrumented code only
Test execution	Modified to pass more options	Modified to add jcov code to the classpath (**)
Performance	Slows the classloading	Faster

* There are options to limit the instrumented code
** Or inject the classes into the application itself

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

“Network” grabber

demo

- Start the grabber on the network

```
java -jar jcov.jar Grabber … -hostname host123 -port 3333 \
  -t template.xml
```

- Run the tests

```
java -classpath ... \
  -javaagent:jcov.jar=grabber,host=host123,port=3333 \
  ...
```

- Stop the grabber – a data file is generated

```
java -jar jcov.jar GrabberManager -kill
```

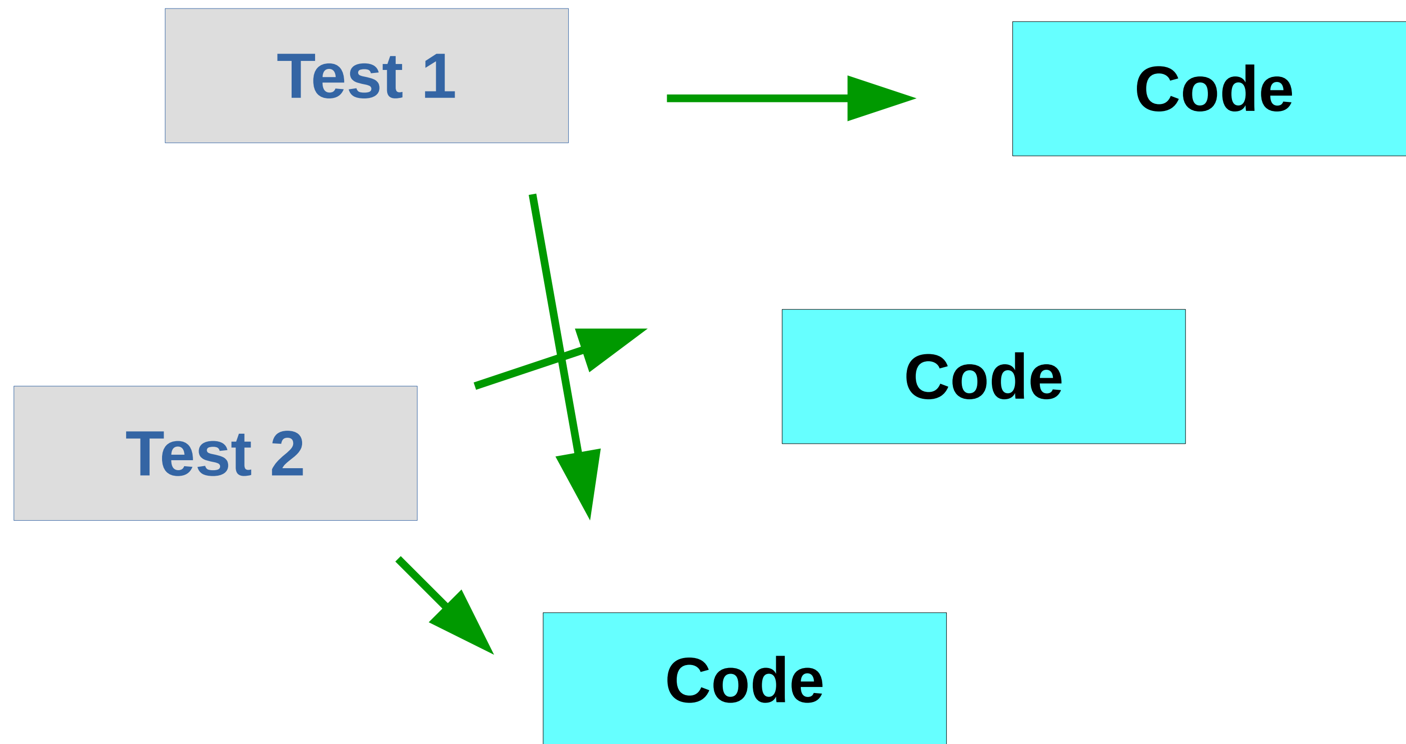
- Merge, generate report

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Individual code coverage

- Track code coverage on a per-test level
- “Usefulness” of the tests



JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Individual test coverage with JCov

- Data is encoded into the XML file:

```
<class name="Component" supername="java/lang/Object" ...>
...
  <meth name="&lt;init&gt;" vmsig="()V" ...>
    <bl s="0" e="151">
      <methenter s="0" e="151" id="20" count="90359"
scale="08ffffffffee7d8efffffffffffffffffffffffffffffffffffffffffffffffff
fffff3effffeffffffffffffffffffffffffffffffffffffffefffbfffffffffffffffbf
fffffffffffffffffd83fffffffffffffffffffffff1"/>
      <exit s="151" e="151" opcode="return"/>
    </bl>
  </meth>
...
</class>
```

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Test scales with files *demo*

- Run tests separately, save **`result.xml`**'s

```
$ java -classpath ... -javaagent:jcov.jar ... my.tests.TestN
$ cp result.xml result_TestN.xml
```

- Merge data files

```
$ java -jar jcov.jar Merger -o merge.xml \
  -outTestList testlist.txt \
  -t template.xml result_*.xml
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Test scales with grabber

demo

- Start the grabber

```
$ java -jar jcov.jar -scale -outTestList testlist.txt\
  -t template.xml
```

- Run tests separately, let the grabber know test names

```
$ java -classpath ... -javaagent:jcov.jar:grabber \
  -Djcov.testname=TestN ... my.tests.TestN
```

- Stop the grabber, generate report.

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Field coverage

demo

- Generate template

```
java -jar jcov.jar TmplGen -field on -t template.xml <in>
```

- Instrumentation

```
ldc             // int 1951
invokestatic   \
  // Method com/sun/tdk/jcov/runtime/CollectDetect.invokeHit:(I)
getfield       // Field x:I
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Abstract API coverage

demo

- Generate template

```
java -jar jcov.jar TmplGen -abstract on -t template.xml <in>
```

- Run with abstract

```
java -classpath ... -javaagent:jcov.jar=abstract=on ...
```

- Instrumentation

```
ldc              // int 3012
invokestatic   \
  // Method com/sun/tdk/jcov/runtime/CollectDetect.invokeHit:(I)
invokeinterface // InterfaceMethod ...
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Save points

demo

- Instrument

```
java -jar jcov.jar Instr -savebegin <a method name> \
   -t template.xml <classes>
```

- Instrumentation for the method:

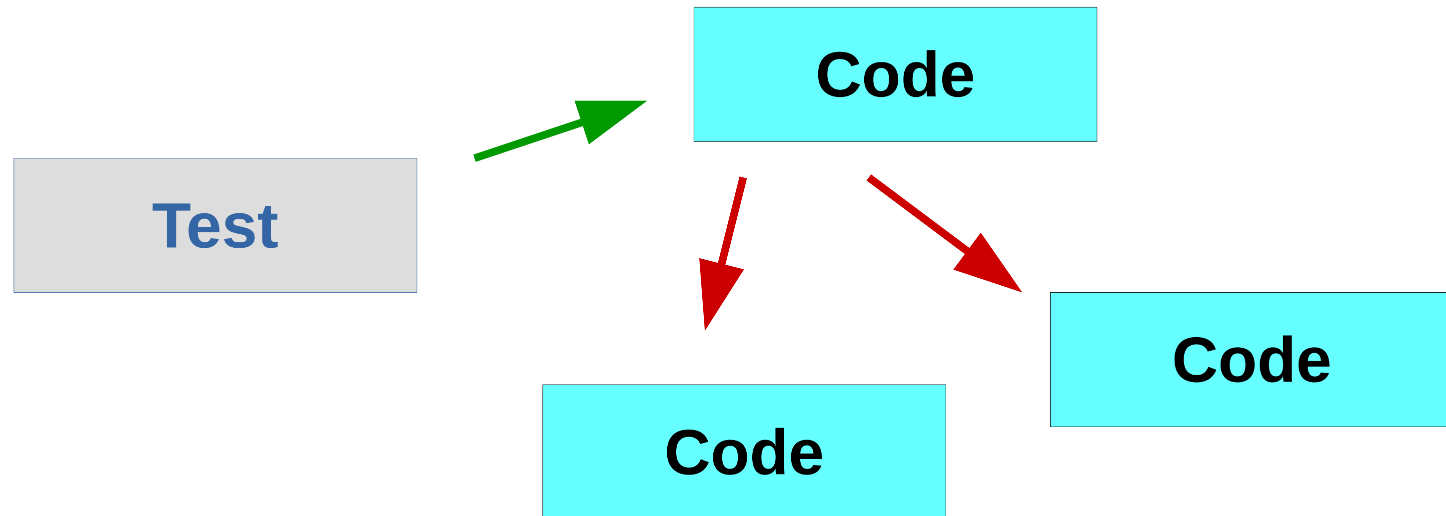
```
0: ldc                  // int 14
2: invokestatic   \
  // Method com/sun/tdk/jcov/runtime/Collect.hit:(I)V
5: invokestatic   \
  // Method com/sun/tdk/jcov/runtime/Collect.saveResults:()V
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Direct coverage

- Only report code called directly from tests
- “Fair” coverage
  - Controlled environment
  - Meaningful parameters



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Direct coverage with no "inner invocation"

demo

- Instrument code only (not tests)

```
java -jar jcov.jar Instr -type method \
    -innerinvocation off \
   -t template.xml <classes>
```

- Instrumentation

```
ldc              // int -1
invokestatic   // Method \
  com/sun/tdk/jcov/runtime/CollectDetect.setExpected:(I)V
... // Some other code
ldc              // int 0
invokestatic   // Method \
  com/sun/tdk/jcov/runtime/CollectDetect.setExpected:(I)V
return
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Direct coverage with "caller include" *demo*

- Run tests, instrumenting code and the tests

```
java -classpath ...\
  -javaagent:jcov.jar=ci=<test packages>,type=method \
  ...
```

- Instrumentation of test code

```
invokestatic \
  // Method com/sun/tdk/jcov/runtime/CollectDetect.setExpected:(I)V
invokevirtual // some product code call
```

- Instrumention of product code

```
invokestatic  \
  // Method com/sun/tdk/jcov/runtime/CollectDetect.hit:(III)V
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Collecting data for big suites

- Decide on dynamic vs static
- Save data periodically
  - Even in case of same vm
  - Otherwise risking loosing the data
- Using file
  - One file for all tests – getting bigger in time – huge performance impact, if multiple VMs
  - Separate files for tests – takes a lot of time to merge
- Use the grabber!
  - Run test consequently for individual test coverage.

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Getting the data

- Instrument dynamically or statically
- Save the data onto a file or grabber
- Individual test coverage
- Abstract coverage
- Fields coverage
- Data save points
- “Direct” coverage

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There is more.

Exec	Executes a command collecting coverage data in a grabb
Agent	print help on usage jcov in dynamic mode
Instr	instruments classfiles and creates template.xml
JREInstr	instrumenter designed for instumenting rt.jar
ProductInstr	
Instr2	instrumenter designed for abstract, native methods anc
TmplGen	generates the jcov template.xml
Grabber	gathers information from JCov runtime via sockets
GrabberManager	control commands to the Grabber server
Merger	merges several jcov data files
RepMerge	merges jcov data files at method level not caring of b
Filter	filters out result data
DiffCoverage	check whether changed lines were covered
RepGen	generates text or HTML (or custom) reports
JCov	gets product coverage with one command

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There is more. TmplGen

```
Verbosity: -verbose
Template specification: -template(t) 'string value'
Type of template: -type [all|branch|block|method]
Filtering conditions:
    -include(i) 'string value', -exclude(e) 'string value',
    -include_list 'string value', -exclude_list 'string value'
Specify which items should be additionally included in template:
    -abstract [on|off], -native [on|off], -field [on|off]
    -synthetic [on|off], -anonym [on|off]
Flush instrumented classes: -flush 'string value'
Basic options: -help(h, ?), -help-verbose(hv)
    -print-env(env), -propfile 'string value'
    -plugindir 'string value', -log.file 'string value'
     -log.level(log) 'string value'
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There is more. Instr

Output: -instr.output(output, o) 'string value'
Verbose mode: -verbose
Type of template: -type [all|branch|block|method]
Filtering conditions: -include(i) 'string value', -include_list 'string value', -exclude(e) 'string value', -caller_include(ci) 'string value', -caller_exclude(ce) 'string value', -exclude_list 'string value'
Save points: -savebegin 'string value', -saveatend 'string value'
Template specification: -template(t) 'string value', -subsequent
Items to be included: -abstract [on|off], -native [on|off], -field [on|off], -synthetic [on|off], -anonym [on|off], -innerinvocation [on|off]
Flush instrumented classes: -flush 'string value'
Runtime management: -implantrt(rt) 'string value', -recursive
Basic options: -help(h, ?), -help-verbose(hv), -print-env(env), -propfile 'string value', -plugindir 'string value', -log.file 'string value', -log.level(log) 'string value'

JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There is more. Merger.

```
Output file: -merger.output(output, o) 'string value'
File to read jcov input files from: -filelist 'string value'
Filtering conditions: -include(i) 'string value'
    -exclude(e) 'string value', -fm 'string value'
    -include_list 'string value', -exclude_list 'string value'
    -fm_list 'string value'
Process/generate test scales: -scale, -outTestList 'string value'
Verbose mode: -verbose(v)
Looseness level:  -loose [0|1|2|3|blocks]
Compress test scales: -compress
Break on error: -breakonerror [file|error|test|skip|none], -critwarn
Template path: -template(tmpl, t) 'string value'
Skipped files: -outSkipped 'string value'
Basic options: -help(h, ?) -help-verbose(hv) -print-env(env)
    -propfile 'string value' -plugindir 'string value
    -log.file 'string value' -log.level(log) 'string value
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Program Agenda

1. History, facts
2. Getting the data
3. **Applied to OpenJDK**
4. Using the data
5. Links, more information

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

OpenJDK coverage. Very simple *demo*

- Template

```
java -jar jcov.jar TmplGen -t template.xml rt.jar
```

- Start grabber

```
java -jar jcov.jar Grabber -v -t template.xml start
```

- Execute tests

```
jtreg ... -javaoptions:"-javaagent:$JCOV_JAR=grabber" <tests>
```

- Stop grabber, merge, generate report

```
java -jar jcov.jar GrabberManager -kill
java -jar jcov.jar Merger -o merged.xml -t template.xml \
  result.xml
java -jar jcov.jar RepGen -src jdk/src/share/classes merged.xml
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

OpenJDK coverage. Static.

- Instrument

```
java -jar jcov.jar JREInstr ... -implant jcov.jar \
  -t template.xml .../jre
```

- Run tests, etc

JavaOne™ ORACLE
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Program Agenda

1. History, facts
2. Getting the data
3. Applied to OpenJDK
4. Using the data
5. Links, more information

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Using the data

- 100% coverage. Needed? Possible? Enough?
- Prioritizing
- Public API
- Monitoring coverage
- Speed up test execution
- More on test development prioritization
- Comparing suites, runs

JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Block coverage target value

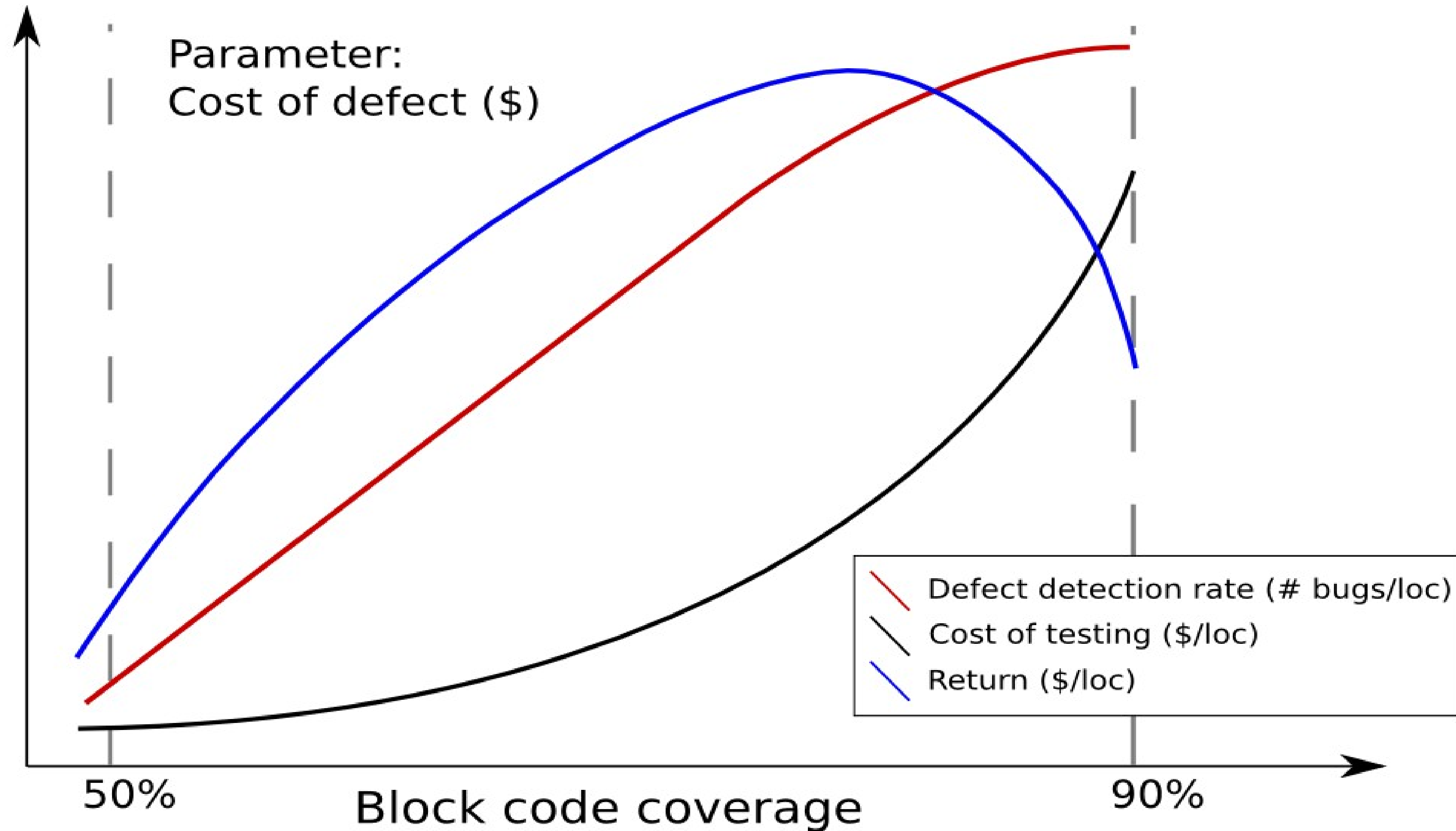
Cost of testing

-

Defects found * COD

=

Return



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

100% block coverage

```
//checks whether the value is zero
public static boolean isZero(float number) {
    boolean result = false;
    if(number != 0) {
        result = false;
    }
    return result;
}
```

1

false

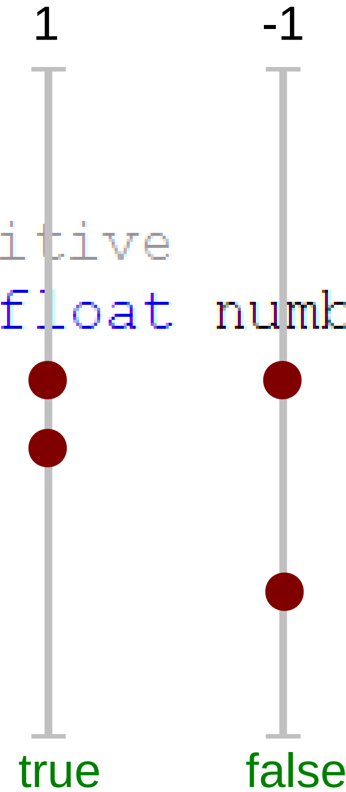
- 1 test
- 100% pass rate
- 100% coverage.
- The bug is not discovered

JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

100% block and branch coverage

```
//checks whether the value is positive
public static boolean isPositive(float number) {
    if(number >= .1) {
        return true;
    }
    return false;
}
```



- 2 tests
- 100% pass rate
- 100% block coverage. 100% branch coverage.
- The bug is not discovered

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Prioritizing test development

- 100% coverage is not the goal
- Too much code to choose from
- Filter the coverage data to leave only code to cover
  - Public API
  - UI
  - Controller code (as in MVC)
- Prioritize code
  - By age
  - By complexity
  - By bug density

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Public API

```
aList.iterator().
```

equals(Object o)	boolean
getClass()	Class<?>
hasNext()	boolean
hashCode()	int
next()	Object
notify()	void
notifyAll()	void
remove()	void

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Public API

- “Methods which supposed to be called directly from user code”
- Every method in public API needs to be called at least once (*)
- 100% public API does not prove anything
- Necessary. Not sufficient.
- Rather blunt solution:

```
java -jar jcov.jar RepGen -publicapi ...
```

  – In JDK8: public and protected methods of public and protected class in java.* and javax.* (**)
  – In JDK9: … something different

** More later*

*** Not completely accurate*

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

JCov API

demo

- `com.sun.tdk.jcov.instrument.`**`DataRoot`**
  - Load, save
- `com.oracle.java.sqe.inspection.`**`JCovFinder, JCovInspector, JCovVisitor`**
  - walk over coverage data
- `com.sun.tdk.jcov.instrument.`**`DataBlock, DataBranch, DataClass, DataMethod, DataPackage`**
  - Investigate, edit
- `com.sun.tdk.jcov.filter.`**`MemberFilter, FilterSPI, Filter`** JCov vommand
  - filter

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Abstract public API

- **Code** coverage
- Good API **is** abstract
- An abstract public API is covered when **at least one** of its implementations is covered.
- Done with JCov abstract coverage and filtering

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

“Public API implementation”

```
aList.iterator().
```

equals(Object o)	boolean
getClass()	Class<?>
hasNext()	boolean
hashCode()	int
next()	Object
notify()	void
notifyAll()	void
remove()	void

```
public class ArrayList<E> implements List<E> {

    //...

    @Override
    public Iterator<E> iterator() {
        return new Itr();
    }

    private class Itr implements Iterator<E> {
```

JavaOne ORACLE

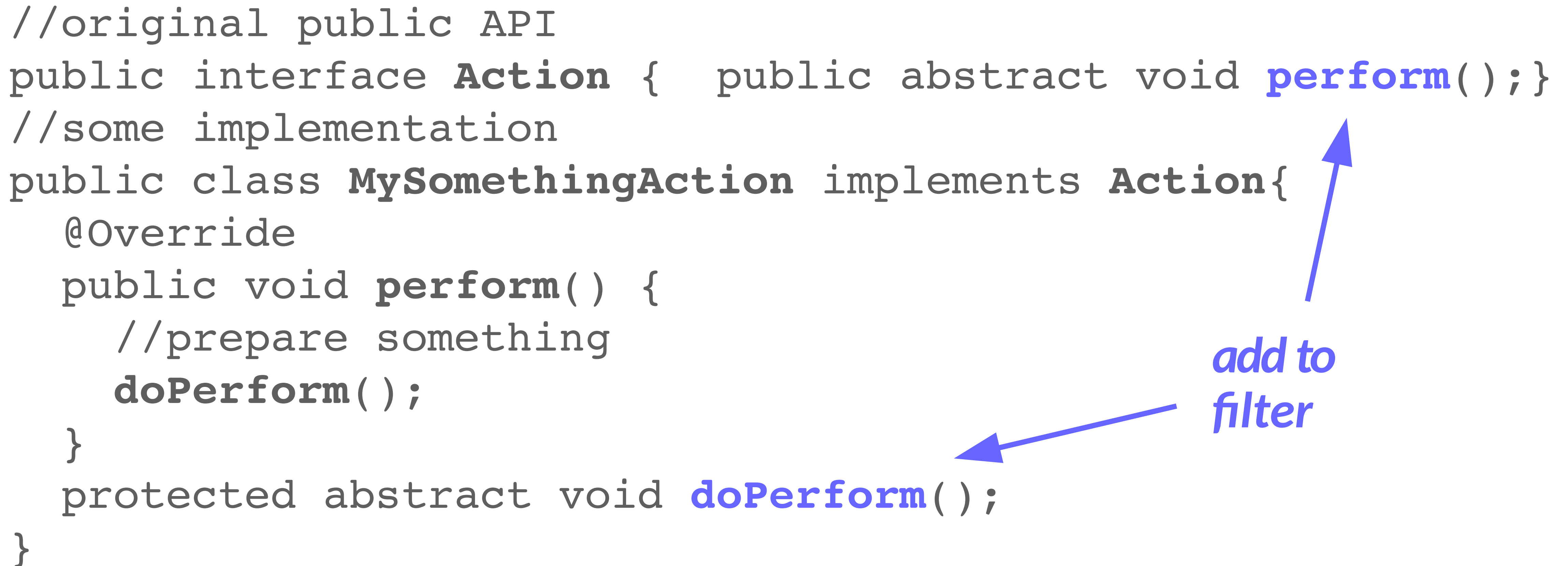
Public API implementation coverage

- Create template with abstract API
- Filter, only leaving
  - Implementations of public API
  - Extensions of public API
- Filter coverage data by the template

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Public API implementation closure



JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Public API closure

demo

- A set of methods implementing library interface
  - Non-abstract public API
  - Implementations of abstract public API
  - Overrides of non-abstract public API
  - “Delegation” API
    - closure of the delegation API

JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Sane public API

demo

"Every method in public API needs to be called at least once" - *Is this so?*

- Some code is trivial
  - One line getters
  - Overloads
- Different code requires different techniques.
  - `java.lang.Object.hashCode()`
- Some code has low impact
  - Exception constructors
- More

JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

UI code coverage

- Similar to public API:
  - If there is a form in the product UI, it needs to be covered at least once
- Identify UI code
  - constructing UI objects
  - displaying UI objects
  - actions with UI
    - javax.sing.Action.actionPerformed

JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Controller code coverage

- Per MVC
  - Controller accepts inputs and and converts to commands to view or model
- Very little boilerplate code
- In some cases identifiable by classes and packages
- Could be marked explicitly. `@Important`

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Linking CC to other characteristics of the source code

Characteristic	Reasoning
Age	New code is better to be tested before getting to customer. Old code is either already tested or not needed. :)
Number of changes	More times the code was changed, more atomic improvements were implemented
Bug density	Many bugs already found means there are more hidden by existing bugs and more could be introduced with fixes
Complexity	Assuming similar engineering talent and the same technology … more bugs are likely to exist in complex code

JavaOne™ ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Linking CC to other characteristics of the source code

- A formula $(1 - cc) * (a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + ...)$
  - cc – code coverage (0, 1)
  - x_i - characteristic
  - a_i – importance coefficient
- Coefficients are important

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Code coverage monitoring

- Build by build
- Platform to platform
- JCov comparison report
- Homegrown database solution (*)
- Requires an apparatus to compare the coverage data

*** More at the end of the presentation***

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Compare two run on the same build

demo

- Comparison report

```
java -jar jcov.jar RepGen file1.xml file2m.xml
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Compare two runs of different builds

demo

- Bytecode is not comparable.
- Loose the details

```
java -jar jcov.jar Merger -loose blocks \
  file1.xml file2m.xml
```

- Only methods left

```
<meth name="isInvalid" vmsig="()Z" ... id="18" count="150"/>
<meth name="getPrefixLength" vmsig="()I" ... id="30" count="28"/>
```

Comparison report

```
java -jar jcov.jar RepGen file1.xml file2m.xml
```

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Subtraction

demo

- Compare two suites A and B
- In ($cc_a - cc_b$) only that code which is covered in cc_a and not in cc_b
- Using JCov API
  - Load cc_a, cc_b
  - Walk over cc_a
  - Search in cc_b
  - If found – null the coverage
- Review ($cc_a - cc_b$) and ($cc_b - cc_a$)

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Test base reduction

- Collect coverage for tests individually and together.
- Compare coverage from every test to the combined coverage.
  - if a test does not add coverage, execute it less
- Conscious choice.
  - Remember that code coverage proves nothing.
  - Only use for acceptance test cycles.
  - Do not throw tests away – run every test although less often.
  - Analyze tests with give no additional coverage.

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Using the data

- 100% coverage. Needed? Possible? Enough?
- Getting coverage subsets to cover 100%
  - Public API
  - UI
  - Controller
- Monitoring coverage
- Test base reduction
- Prioritizing test dev by ranking
- Comparing suites, runs

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

More info

- Wiki: https://wiki.openjdk.java.net/display/CodeTools/jcov
- Source: http://hg.openjdk.java.net/code-tools/jcov
- Tutorial: http://hg.openjdk.java.net/code-tools/jcov/raw-file/tip/examples/tuto
- How to build JCov: https://wiki.openjdk.java.net/display/CodeTools/How+To+Bu

- "Pragmatic code coverage" on slideshare.net

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

More to come

- Plugins
  - IDE: Netbeans, Eclipse, Intellij Idea
  - maven
- Backend
  - Coverage storage database
  - Trends, statistics
  - Ranking
- Test base reduction

Hey, it's open-source! Contribute.

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Java Code Coverage with JCov

Implementation Details and Use Cases

Alexandre (Shura) Iline
JDK Test Architect
JDK SQE team
Sept 1, 2014

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.