Coverage tools

- Program is typically compiled with special options, to add extra source or object code.
  - Additional data structures, such as a flow graph, may also be created.
- Program is run, possibly via test cases
  - During execution, information is accumulated and written to an output file.
- Post-processing phase:
  - User report is generated from output file.

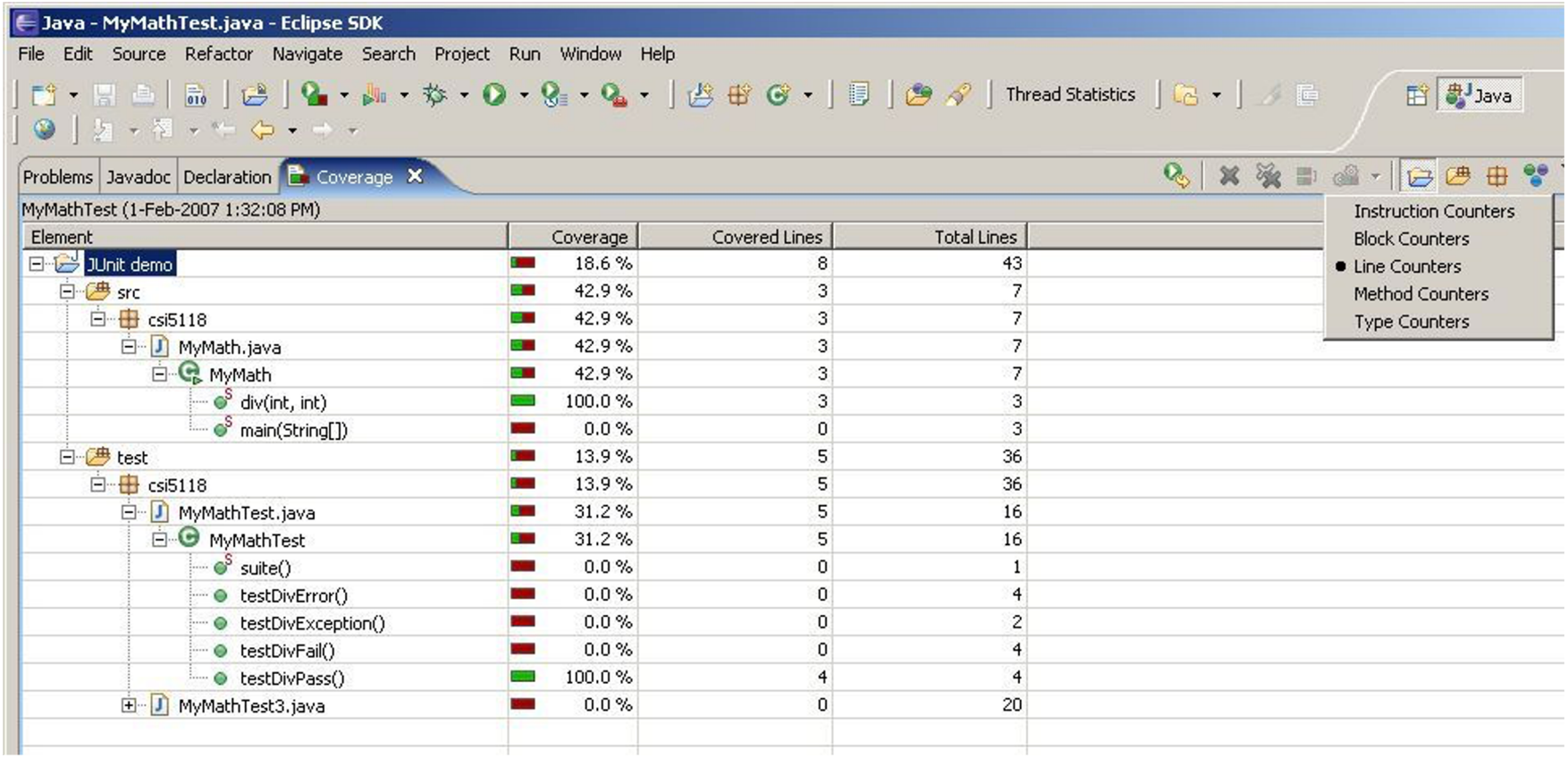
Java Code Coverage Library (JaCoCo)

- Open-source tool
  - Supports class, method, "basic block", and line coverage.
  - "Fractional" line coverage supported, but not branch coverage.
  - Standalone version works with Ant builds
  - http://emma.sourceforge.net
  - Eclipse plugin EclEmma also available
  - http://www.eclemma.org

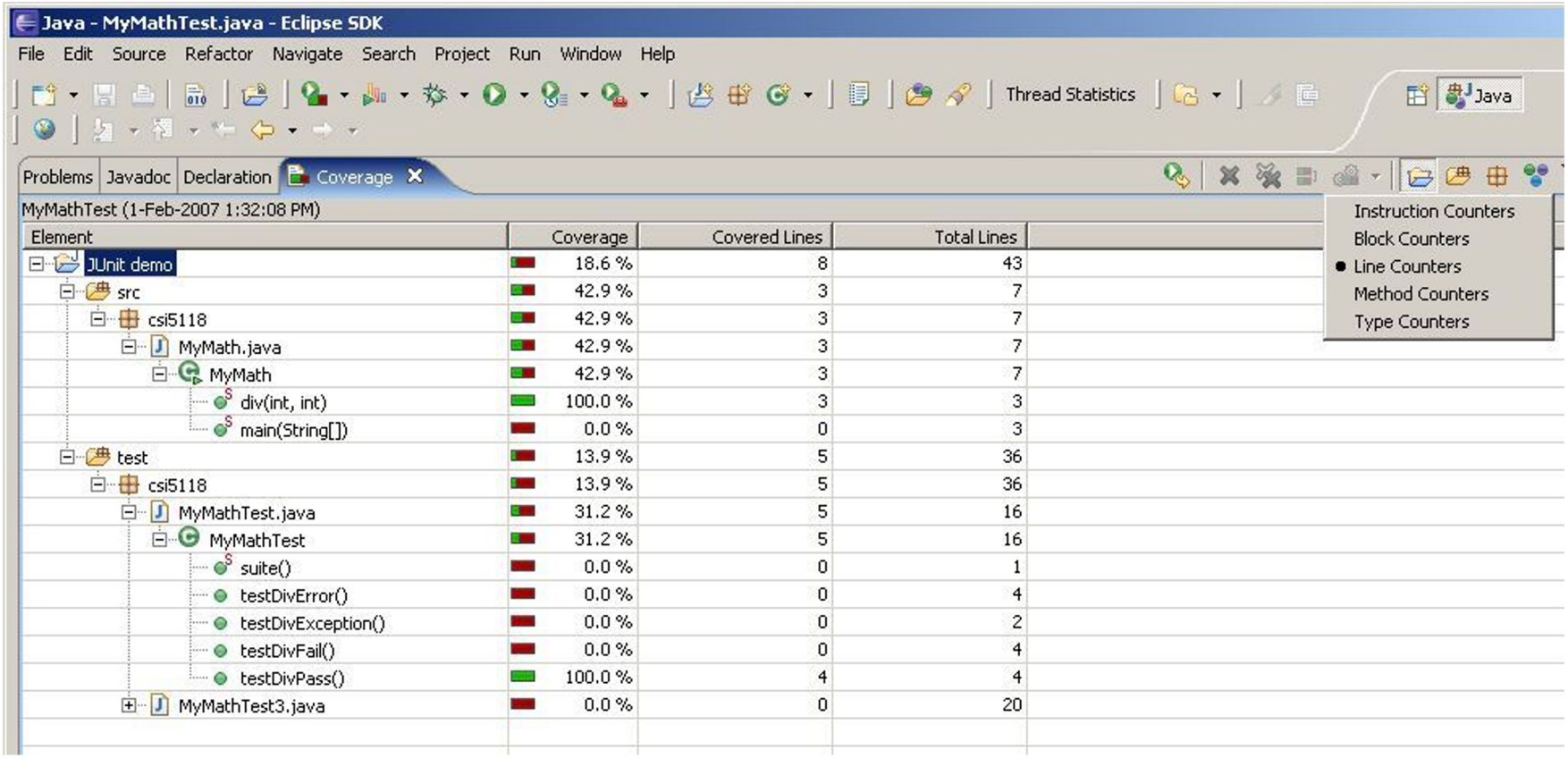
Block Coverage

- Block coverage assumes that if a block of statements without branches is entered and exited, all statements in the block were executed.
  - That is, the counter is at the end of the block, instead of before the source code statement.
- Result: If an exception occurs in the block, the entire block is not recorded as having executed.
  - This may be fine for application source code, but it does not work well with JUnit test source code or in code for which exceptions are commonplace.
  - JUnit throws exceptions internally when tests fail, so the test may not have appeared to be executed.

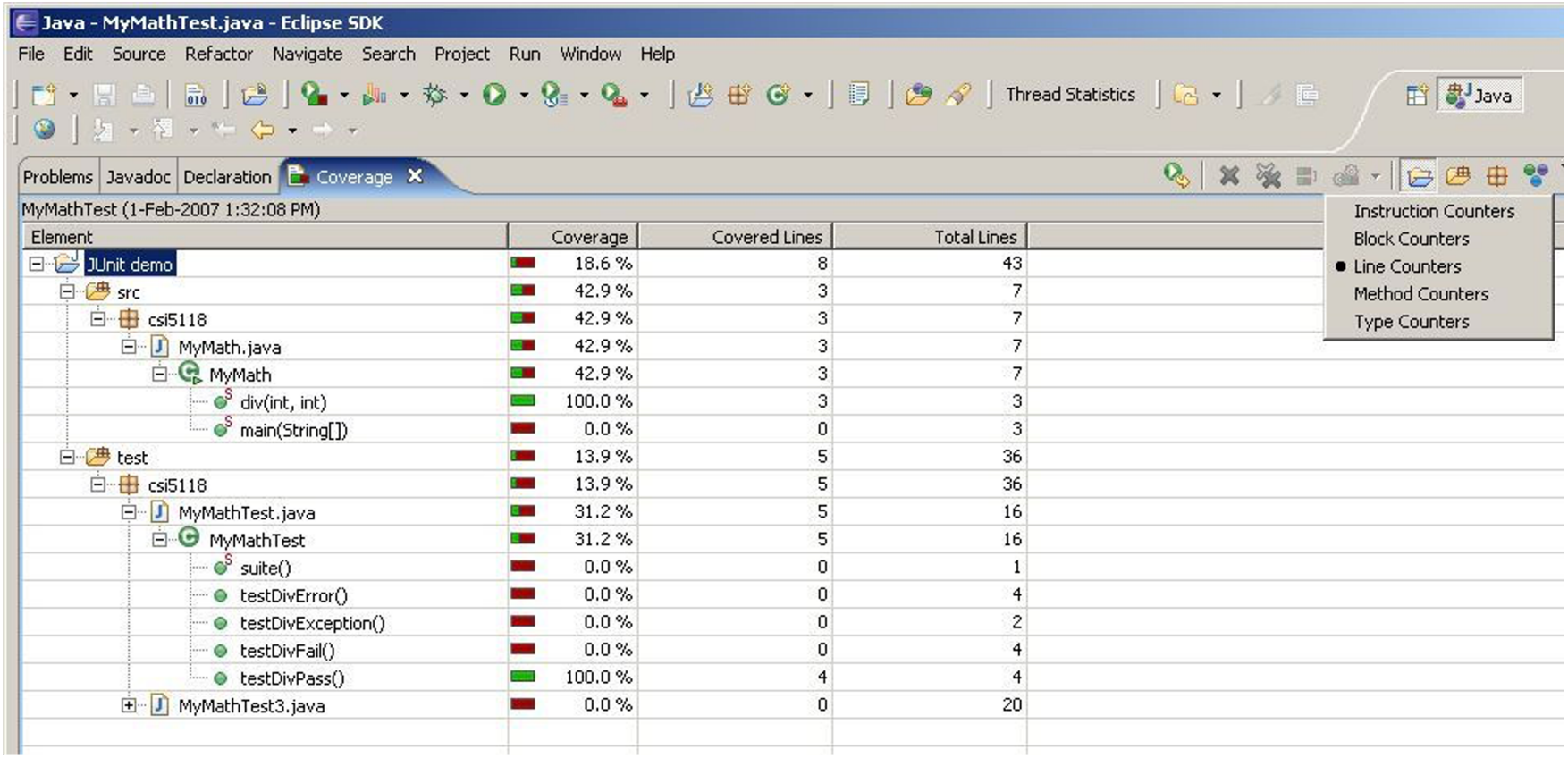
Emma coverage report



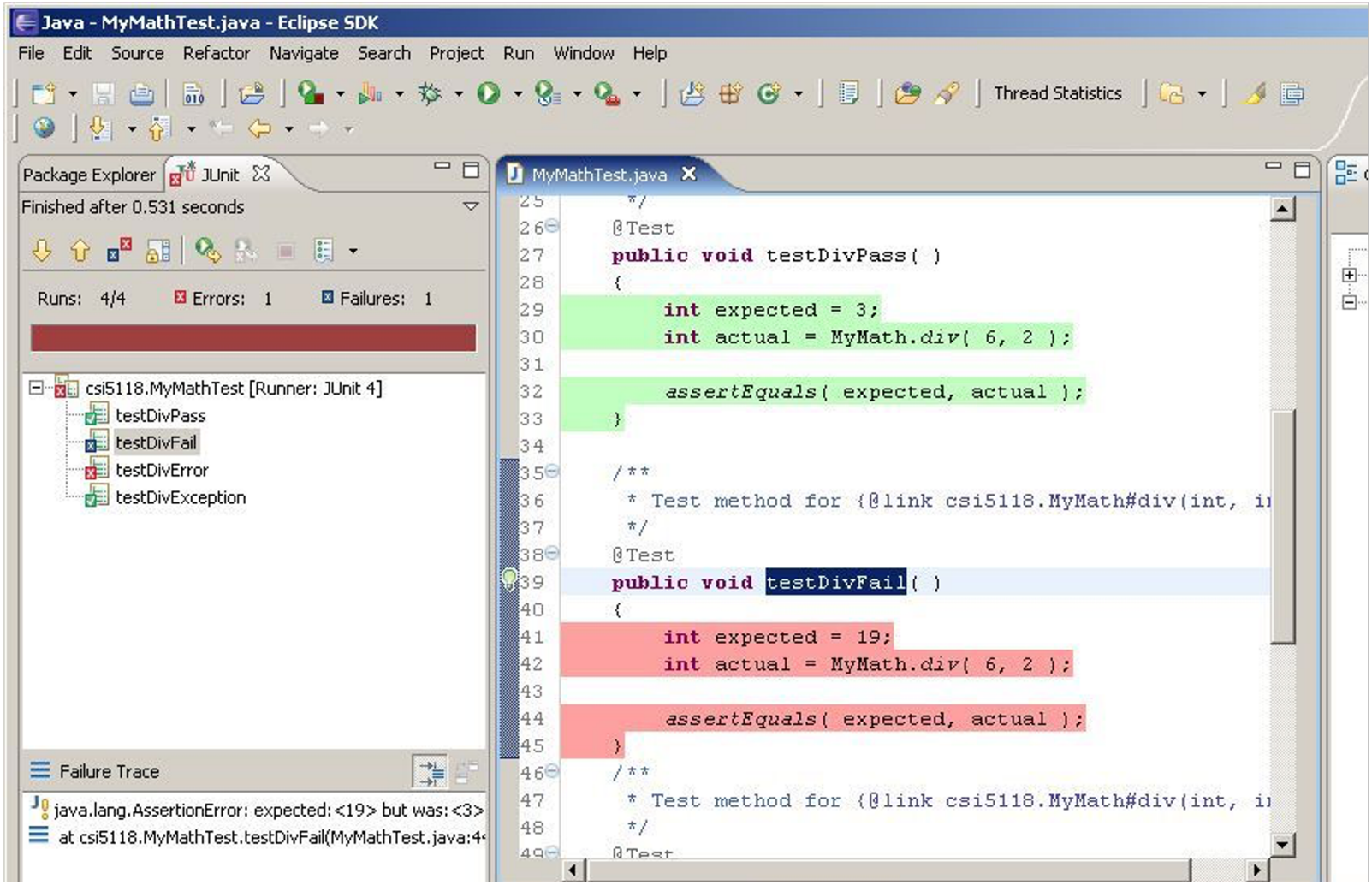
Emma coverage report



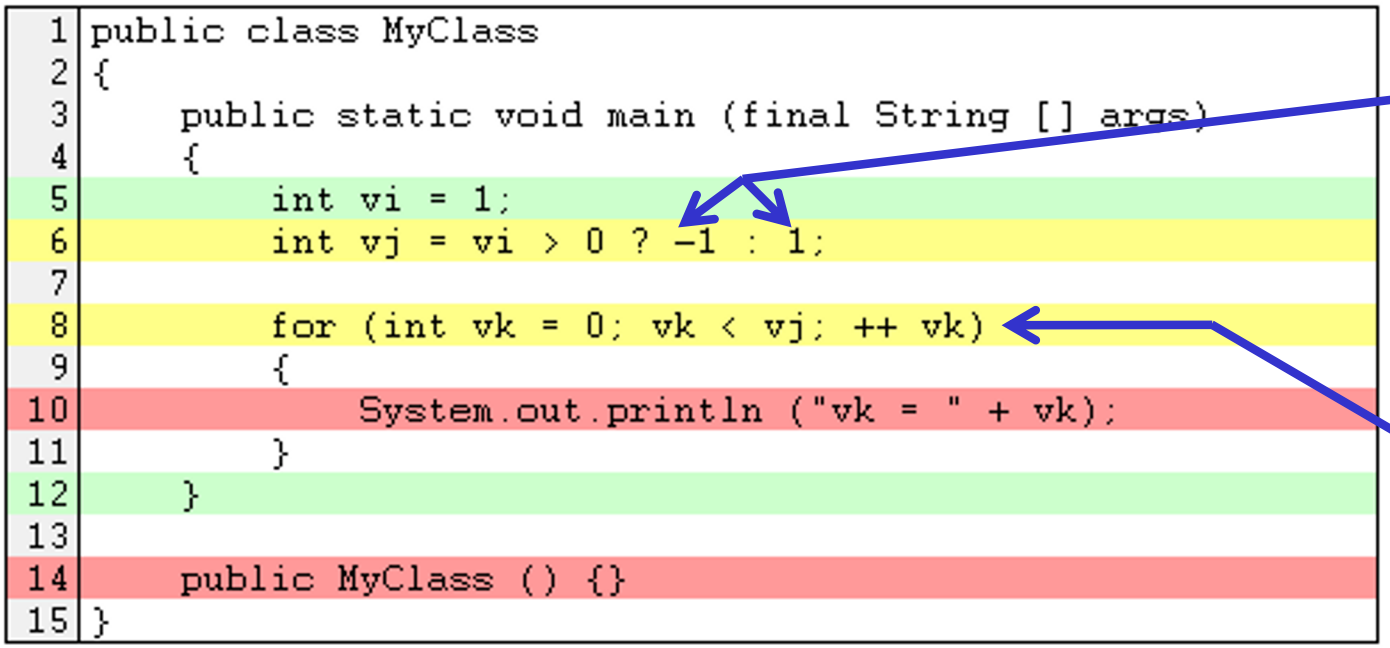
Emma coverage report



Emma source code annotations



Fractional line coverage



```
 1 public class MyClass
 2 {
 3     public static void main (final String [] args)
 4     {
 5         int vi = 1;
 6         int vj = vi > 0 ? -1 : 1;
 7
 8         for (int vk = 0; vk < vj; ++ vk)
 9         {
10             System.out.println ("vk = " + vk);
11         }
12     }
13
14     public MyClass () {}
15 }
```

CodeCover

- Open source Eclipse plug in
  - Web site: http://codecover.org

- Performs source-code instrumentation to obtain:
  - Statement coverage
  - Branch coverage
  - Loop coverage: loop executed zero/once/many times.
  - MC/DC Coverage (Term coverage)
  - ?-Operator Coverage
  - Synchronized Operations Coverage

CodeCover summary report

Name	Statement	Branch	Loop	Term	?-Operato	Synchronized
BitTest	16.7 %	33.3 %	–	28.6 %	0.0 %	–
demo	16.7 %	33.3 %	–	28.6 %	0.0 %	–
Bit	16.7 %	33.3 %	–	28.6 %	0.0 %	–
hashCode	0.0 %	–	–	–	0.0 %	–
Bit	0.0 %	–	–	–	–	–
Bit	0.0 %	–	–	–	–	–
Bit	0.0 %	–	–	–	–	–
Bit	100.0 %	–	–	–	–	–
and	100.0 %	–	–	–	–	–
equals	33.3 %	50.0 %	–	50.0 %	–	–
getIntValue	0.0 %	0.0 %	–	0.0 %	–	–
not	0.0 %	–	–	–	–	–
or	0.0 %	–	–	–	–	–
setValue	0.0 %	0.0 %	–	0.0 %	–	–
setValue	0.0 %	–	–	–	–	–
toString	0.0 %	–	–	–	–	–
xor	0.0 %	–	–	–	–	–

CodeCover detailed report

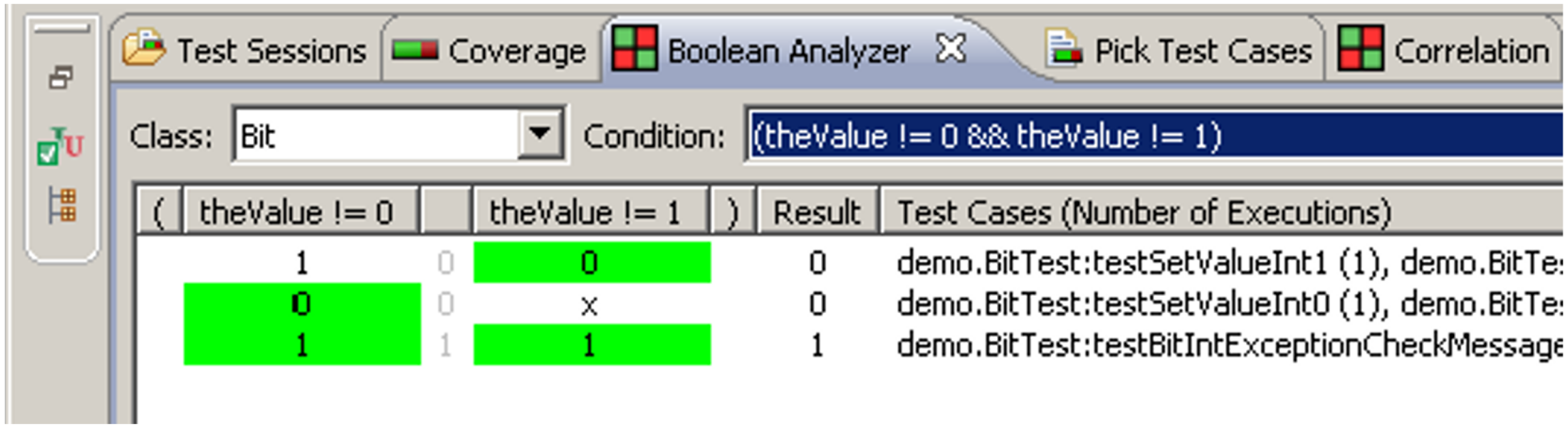
```
191         // Identity check
192
193         if ( this == obj )
194             return true;
195
196         // Null operand check
197
198         if ( obj == null )
199             return false;
200
201         // Type of objects must be the same
202
203         if ( getClass( ) != obj.getClass( ) )
204             return false;
205
206         // We now know there are two distinct Bit objects.  Compare their values.
207
208         final Bit other = ( Bit ) obj;
209         if ( value != other.value )
210             return false;
211         return true;
212     }
```

Unexecuted branches: then

CodeCover features

- Boolean value analyzer: shows how many Boolean combinations in conditions have been covered.
- Code "hot spots": highlighting of code that is executed more frequently than most.
- Test correlation matrix: for each pair of test cases, the overlap in coverage for the two test cases is shown.

CodeCover Boolean analyzer



- For the compound condition shown, combinations of atomic conditions that have occurred are shown.
  - The **x** shows a short-circuit evaluation.

CodeCover Correlation view

- The colours give an indication of the overlap for pairs of test cases.
  - The selected square shows that for these two test cases, they have 24 and 25 coverable items, and that 19 are shared, for an overlap of 88.1%

References

- Emma:
  - http://emma.sourceforge.net
  - http://www.eclemma.org

- CodeCover: http://codecover.org

- A. Glover, "Don't be fooled by the Coverage Report", IBM developer works article
  http://www-128.ibm.com/developerworks/java/library/j-cq01316
- S. Gornett, "Code Coverage Analysis"
  - http://www.bullseye.com/coverage.html