

Analizadores estáticos

Raúl Estrada

Octubre 2020

1. Overview

In our [introduction to FindBugs](#), we looked at the functionality of FindBugs as a static analysis tool and how it can be directly integrated into IDEs like Eclipse and IntelliJ Idea.

In this article, we're going to look into a few of the alternative static analysis tools for Java – and how these integrate with Eclipse and IntelliJ IDEA.

2. PMD

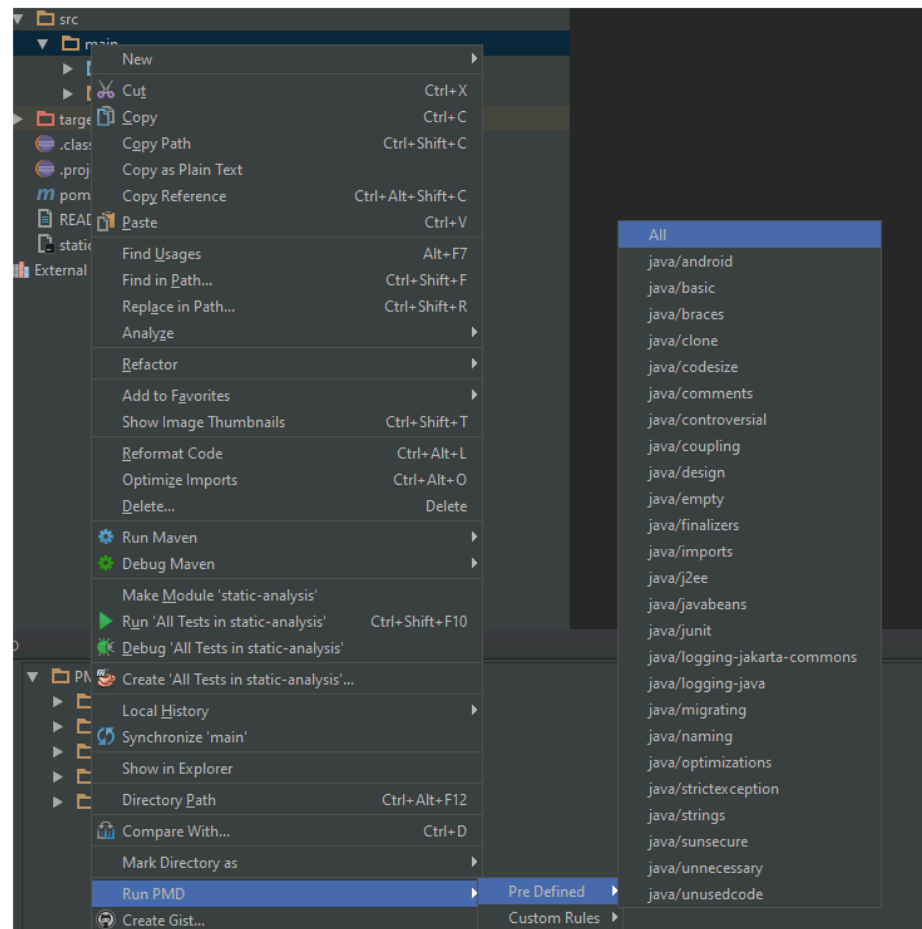
Let's start with PMD.

This mature and quite well-established tool analyzes source code for possible bugs, suboptimal codes and other bad practices; it also looks at more advanced metrics such as cyclomatic complexity for the codebase it analyzes.

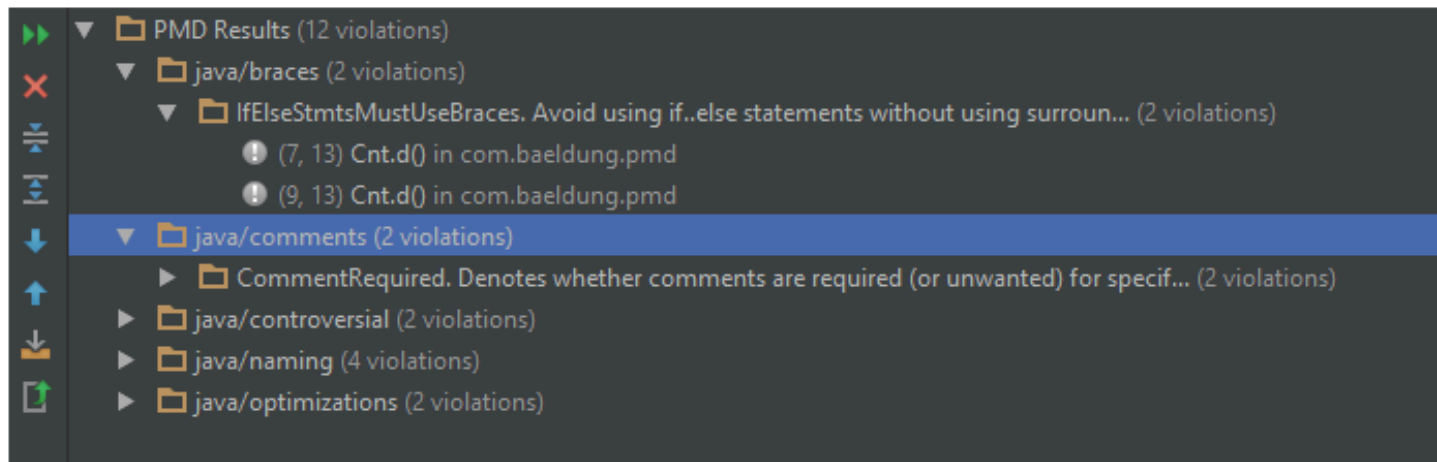
2.2. Integration With IntelliJ

Of course, IntelliJ has a similar PMD plugin – which can be downloaded and installed from the [JetBrains plugin store](#).

We can similarly run the plugin right in the IDE – by right-clicking the source we need to scan and selecting PMD scan from the context menu:



Results are displayed immediately but, unlike in Eclipse, if we try to open the description it will open up a browser with a public web page on finding information:



We can set the behavior of the PMD plugin from the settings page, by going to File -> Settings -> other settings -> PMD to view configuration page. From the settings page, we can configure the rule set by loading a custom rule set with our own testing rules.

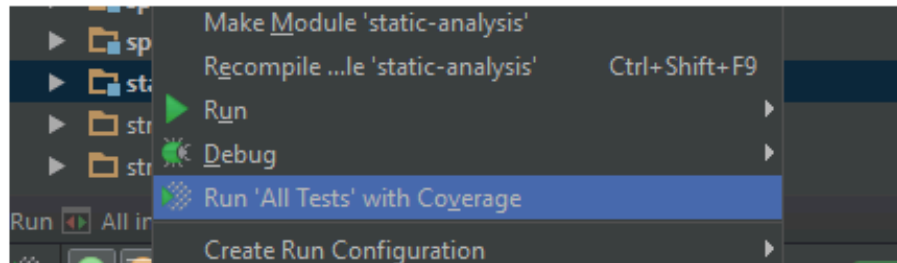
3. JaCoCo

Moving on – JaCoCo is a test coverage tool – used to keep track of unit test coverage in the codebase. Simply put, the tool calculates the coverage using a number of strategies e.g.: lines, class, methods, etc.

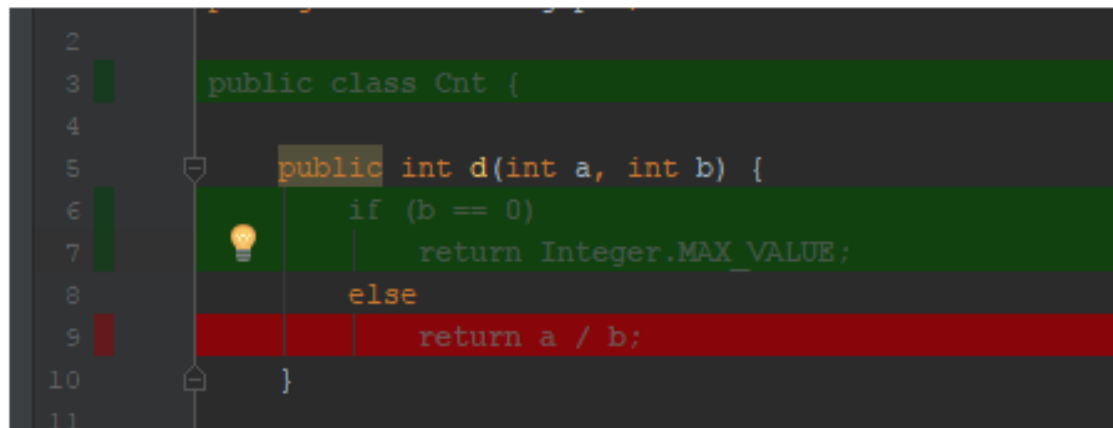
3.2. Integration With IntelliJ IDEA

JaCoCo is bundled by default with the latest IntelliJ IDEA distribution, so there's no requirement to install the plugin separately.

When executing unit tests, we can select what coverage runner we need to use. We can run the test cases either at the project level or at the class level:



Similar to Eclipse, JaCoCo displays results using different color schemes for the coverage.

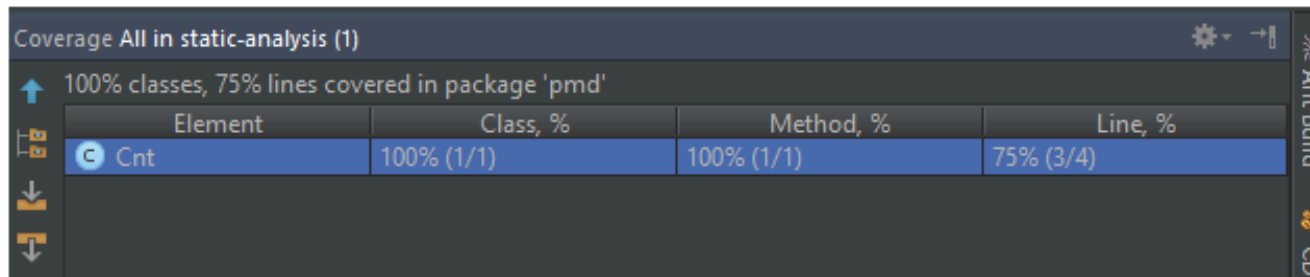


The screenshot shows a code editor with the following Java code:

```
2  
3 public class Cnt {  
4  
5     public int d(int a, int b) {  
6         if (b == 0)  
7             return Integer.MAX_VALUE;  
8         else  
9             return a / b;  
10    }  
11
```

The code is annotated with JaCoCo coverage markers. The line numbers 2 through 11 are on the left. The code is color-coded: the class declaration and the if-statement branch are green, while the else branch is red. A yellow lightbulb icon is placed next to the if-statement branch. A vertical line with a shield icon is positioned between the line numbers and the code, indicating the current execution path.

We can see the summary of the test coverage where it displays how much of the code is covered under unit tests in class level and package levels.



Coverage All in static-analysis (1)

100% classes, 75% lines covered in package 'pmd'

Element	Class, %	Method, %	Line, %
Cnt	100% (1/1)	100% (1/1)	75% (3/4)

Ant Build CD

4. Cobertura

Finally, it's worth mentioning Cobertura – this is similarly used to keep track of unit test coverage in the codebase. The latest version of Eclipse doesn't support the Cobertura plugin at the time of writing; the plugin does work with earlier Eclipse versions.

Similarly, IntelliJ IDEA doesn't have an official plugin which can execute the Cobertura coverage.

5. Conclusion

We looked at integration with Eclipse and IntelliJ IDEA for three widely used static analysis tools. FindBug was covered in a previous [introduction to FindBugs](#).