

Findbugs

Raúl Estrada

Octubre 2020

1. Overview

FindBugs is an open source tool used to perform **static analysis** on Java code.

In this article, we're going to have a look at setting up FindBugs on a Java project and integrating it into the IDE and the Maven build.

2. FindBugs Maven Plugin

2.1. Maven Configuration

In order to start generating static analysis reports, we first need to add the FindBugs plugin in our *pom.xml*

```
1  <reporting>
2    <plugins>
3      <plugin>
4        <groupId>org.codehaus.mojo</groupId>
5        <artifactId>findbugs-maven-plugin</artifactId>
6        <version>3.0.4</version>
7      </plugin>
8    </plugins>
9  </reporting>
```

You can check out the [latest version of the plugin](#) on Maven Central.

2.2. Report Generation

Now that we have the Maven plugin properly configured, let's generate the project documentation using the *mvn site* command.

The report will be generated in the folder **target/site** in the project directory under the name **findbugs.html**.

You can also run the *mvn findbugs:gui* command to launch the GUI interface to browse the generated reports for the current project.

The FindBugs plugin can also be configured to fail under some circumstances – by adding the execution goal *check* to our configuration:

```
1  <plugin>
2    <groupId>org.codehaus.mojo</groupId>
3    <artifactId>findbugs-maven-plugin</artifactId>
4    <version>3.0.4</version>
5    <configuration>
6      <effort>Max</effort>
7    </configuration>
8    <executions>
9      <execution>
10        <goals>
11          <goal>check</goal>
12        </goals>
13      </execution>
14    </executions>
15  </plugin>
```

The *effort* – when maxed out, performs a more complete and precise analysis, revealing more bugs in the code, though, it consumes more resources and takes more time to complete.

You can now run the command *mvn verify*, to check if the build will succeed or not – depending on the defects detected while running the analysis.

You can also enhance the report generation process and take more control over the analysis, by adding some basic configuration to the plugin declaration:

```
1 <configuration>
2   <onlyAnalyze>org.baeldung.web.controller.*</onlyAnalyze>
3   <omitVisitors>FindNullDeref</omitVisitors>
4   <visitors>FindReturnRef</visitors>
5 </configuration>
```

The *onlyAnalyze* option declares a comma separated values of classes/packages eligible for analysis.

The *visitors/omitVisitors* options are also comma separated values, they are used to specify which detectors should/shouldn't be run during the analysis – Note that ***visitors* and *omitVisitors* cannot be used at the same time.**

A detector is specified by its class name, without any package qualification. Find the details of all detectors class names available by following [this link](#).

4. FindBugs IntelliJ IDEA Plugin

4.1. Installation

If you are an IntelliJ IDEA fan, and you want to start inspecting Java code using FindBugs, you can simply grab the plugin installation package from the [official JetBrains site](#), and extract it to the folder %INSTALLATION_DIRECTORY%/plugins. Restart your IDE and you're good to go.













Alternatively, you can navigate to Settings -> Plugins and search all repositories for FindBugs plugin.

By the time of writing this article, the version 1.0.1 of the IntelliJ IDEA plugin is just out,

To make sure that the FindBugs plugin is properly installed, check for the option labeled "Analyze project code" under Analyze -> FindBugs.

4.2. Reports Browsing

In order to launch static analysis in IDEA, click on "Analyze project code", under Analyze -> FindBugs, then look for the FindBugs-IDEA panel to inspect the results:

- ▲  > spring-rest (20) [Baeldung master]
 - ▲  Of Concern (20)
 - ▲  Normal confidence (10)
 - ▷  Method ignores exceptional return value (2)
 - ▷  Unused field (7)
 - ▷  Unwritten public or protected field (1)
 - ▲  Low confidence (10)
 - ▷  Confusing method names (1)
 - ▷  Method may fail to close stream on exception (2)
 - ▷  Exception is caught when Exception is not thrown (2)
 - ▷  Private method is never called (3)
 - ▷  Field not initialized in constructor but dereferenced without null check (2)

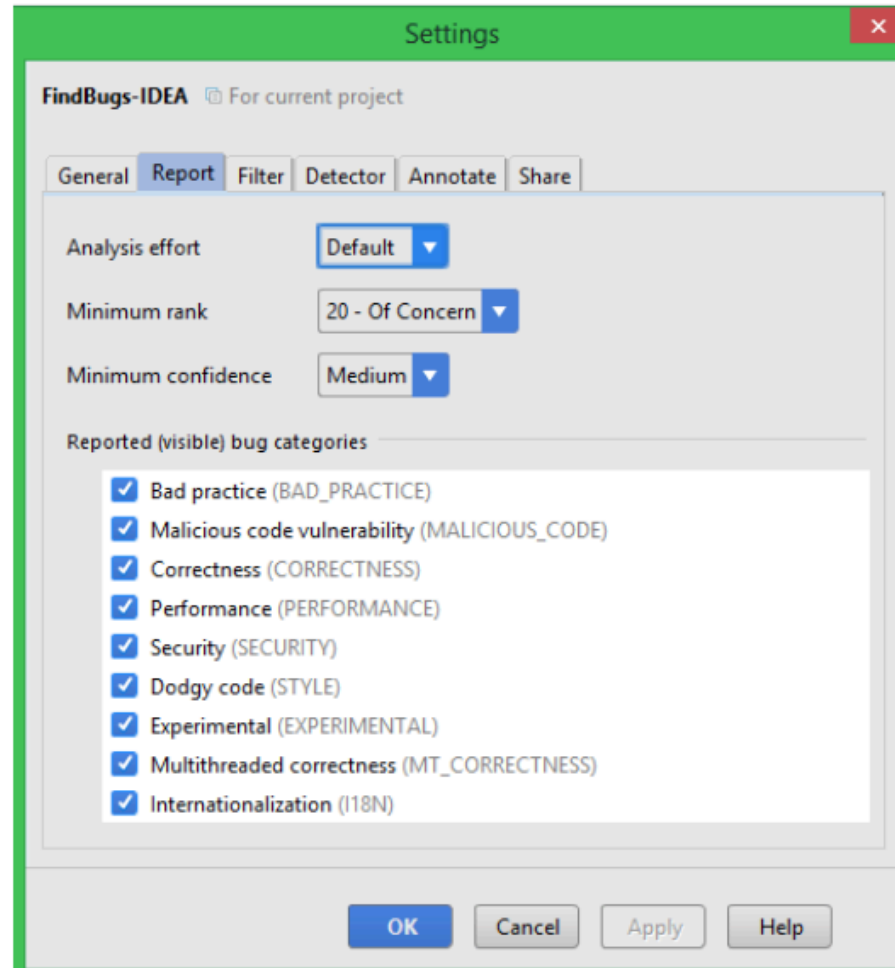
You can use the second column of commands on the left side of the screenshot, to group defects using different factors:

1. Group by a bug category.
2. Group by a class.
3. Group by a package.
4. Group by a bug rank.

It is also possible to export the reports in XML/HTML format, by clicking the "export" button in the fourth column of commands.

4.3. Configuration

The FindBugs plugin preferences pages inside IDEA is pretty self-explanatory:















This settings window is quite similar to the one we've seen in Eclipse, thus you can perform all kinds of configuration in an analogous fashion, starting from analysis effort level, bugs ranking, confidence, classes filtering, etc.

The preferences panel can be accessed inside IDEA, by clicking the "Plugin preferences" icon under the FindBugs-IDEA panel.

5. Report Analysis for the Spring-Rest Project

In this section we're going to shed some light on a static analysis done on the [spring-rest project available on Github](#) as an example:

- ▲  > spring-rest (20) [Baeldung master]
 - ▲  Of Concern (20)
 - ▲  Normal confidence (10)
 - ▷  Method ignores exceptional return value (2)
 - ▷  Unused field (7)
 - ▷  Unwritten public or protected field (1)
 - ▲  Low confidence (10)
 - ▷  Confusing method names (1)
 - ▷  Method may fail to close stream on exception (2)
 - ▷  Exception is caught when Exception is not thrown (2)
 - ▷  Private method is never called (3)
 - ▷  Field not initialized in constructor but dereferenced without null check (2)

Most of the defects are minor — Of Concern, but let's see what we can do to fix some of them.

Method ignores exceptional return value:

```
1 | File fileServer = new File(fileName);  
2 | fileServer.createNewFile();
```

As you can probably guess, FindBugs is complaining about the fact that we're throwing away the return value of the *createNewFile()* method. A possible fix would be to store the returned value in a newly declared variable, then, log something meaningful using the DEBUG log level — e.g. "*The named file does not exist and was successfully created*" if the returned value is true.

The method may fail to close stream on exception: this particular defect illustrates a typical use case for exception handling that suggests to **always close streams in a *finally* block**:

```
1  try {
2      DateFormat dateFormat
3          = new SimpleDateFormat("yyyy_MM_dd_HH.mm.ss");
4      String fileName = dateFormat.format(new Date());
5      File fileServer = new File(fileName);
6      fileServer.createNewFile();
7      byte[] bytes = file.getBytes();
8      BufferedOutputStream stream
9          = new BufferedOutputStream(new FileOutputStream(fileServer));
10     stream.write(bytes);
11     stream.close();
12     return "You successfully uploaded " + username;
13 } catch (Exception e) {
14     return "You failed to upload " + e.getMessage();
15 }
```

When an exception is thrown before the *stream.close()* instruction, the stream is never closed, that's why it's always preferable to make use of the *finally()* block to close streams opened during a *try/catch* routine.

An *Exception* is caught when *Exception* is not thrown: As you may already know, catching *Exception* is a bad coding practice, FindBugs thinks that you must catch a most specific exception, so you can handle it properly. So basically manipulating streams in a Java class, catching *IOException* would be more appropriate than catching a more generic *Exception*.

Field not initialized in the constructor but dereferenced without null check: it's always a good idea to initialize fields inside constructors, otherwise, we should live with the possibility that the code will raise an *NPE*. Thus, it is recommended to perform null checks whenever we're not sure if the variable is properly initialized or not.

6. Conclusion

In this article, we've covered the basic key points to use and customize FindBugs in a Java project.

As you can see, FindBugs is a powerful, yet simple static analysis tool, it helps to detect potential quality holes in your system – if tuned and used correctly.

Finally, it is worth mentioning that FindBugs can also be run as part of a separate continuous automatic code review tool like **Sputnik**, which can be very helpful to give the reports a lot more visibility.