Tutorial & Lab: PMD and FindBugs

Using PMD and FindBugs to write more efficient code

Introduction

Last week we looked at the IntelliJ static analyzer and how it could benefit us in writing better, cleaner, and sometimes faster code without changing what your code actually does. This week we will be exploring two more static analyzers that you should have installed during the pre-class assignment, FindBugs and PMD. It can be beneficial to run through multiple different static analyzers that specialize in different areas to help improve your code. Another benefit of PMD and FindBugs over the built-in Intellij static analyzer is that these tools can be run independently of an IDE, so they can be built into an automated code deployment process. We won't build that kind of a deployment process in this lab, but we wanted you to understand the motivation for learning these tools in addition to one that's built into Intellij.

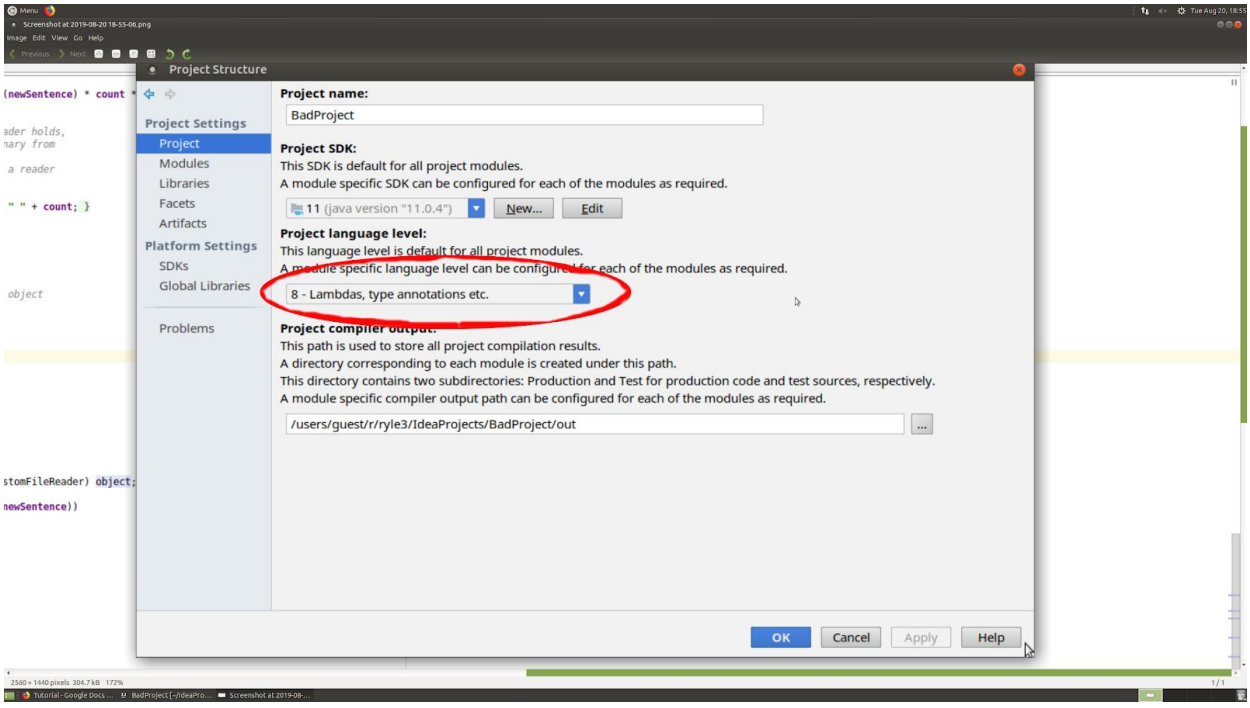
As you examine the java code given to you, you will notice that we are using the same program from last week with a few minor changes. This is to show that even though the IntelliJ static analyzer doesn't show any warnings, there are still other ways to possibly improve our code.

Download the revised code and set up the file structure the same as last week. Remember to run the program once to create a run configuration and then edit the run configuration. Refer to the instructions from the last tutorial for information on how to set up the run configuration. After you ensure that the program is working properly, run IntelliJ's static analyzer. The green bar should display as it did at the end of last week's lab, meaning that it didn't detect any errors.

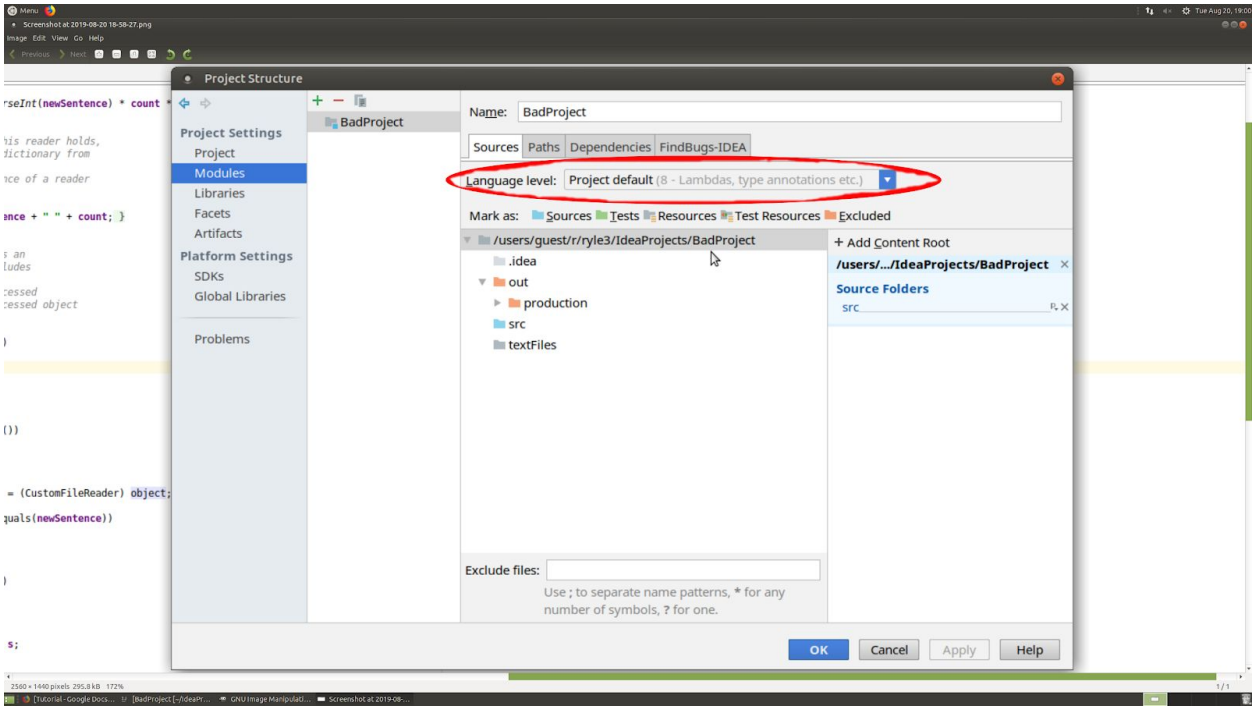
Tutorial

1. Let's start by running FindBugs to see if we can find any possible problems with our code. Since FindBugs is an older program, the newer versions of java are not compatible with it. We will need to change our **Project language level** to one that is compatible with FindBugs in order for it to run properly.

2. First, click on File > Project Structure, and in the "Project Settings" tab to the left, make sure you are in the "Project" view. From here change your **Project language level:** to "8 - Lambdas, type annotations etc.".

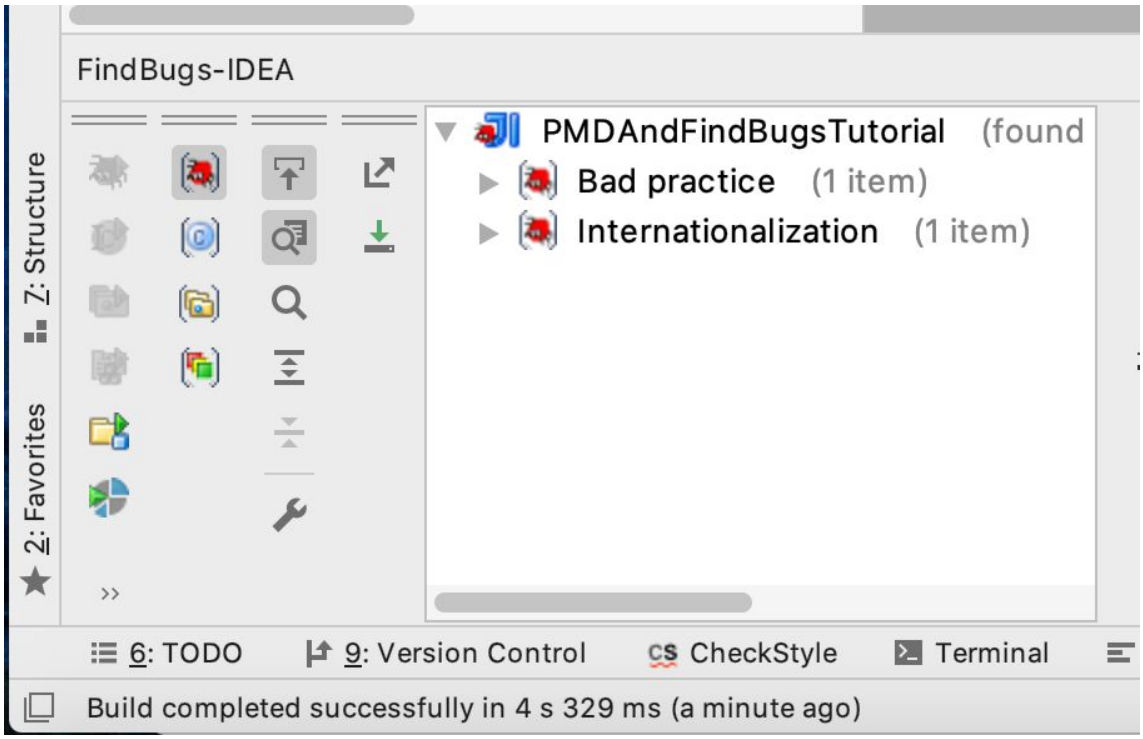


This will change your overall project to that language level. We also need to change the module language to match. On the left side of the window under “Project Settings” click on “Modules”. You should see the name of your project selected and also a drop down bar next to the words “Language level:”. Click on the drop down and set it to the same language we set for our project structure (8 - Lambdas, type annotations, etc). When you are done, click OK. You should now be able to run FindBugs on your project.



3. Right click on the name of your project and click FindBugs > Analyze Project FIles. What should show up first is a window that says “Do you want to include test sources for the analysis?”, to which you should click yes.

4. You should see output similar to the following image in the bottom left corner of your Intellij screen:



It indicates that there are two potential bugs. Open the one that says “Bad Practice” and keep opening until there are no more diamonds to click. It says there is a potential bug in our equals method. The last three methods that you see in the CustomFileReader class have the keyword “@Override” above each declaration. This means it is rewriting a method with the exact same name from the inherited Object class. Overriding these methods for your custom classes can be useful but only if they are implemented correctly.

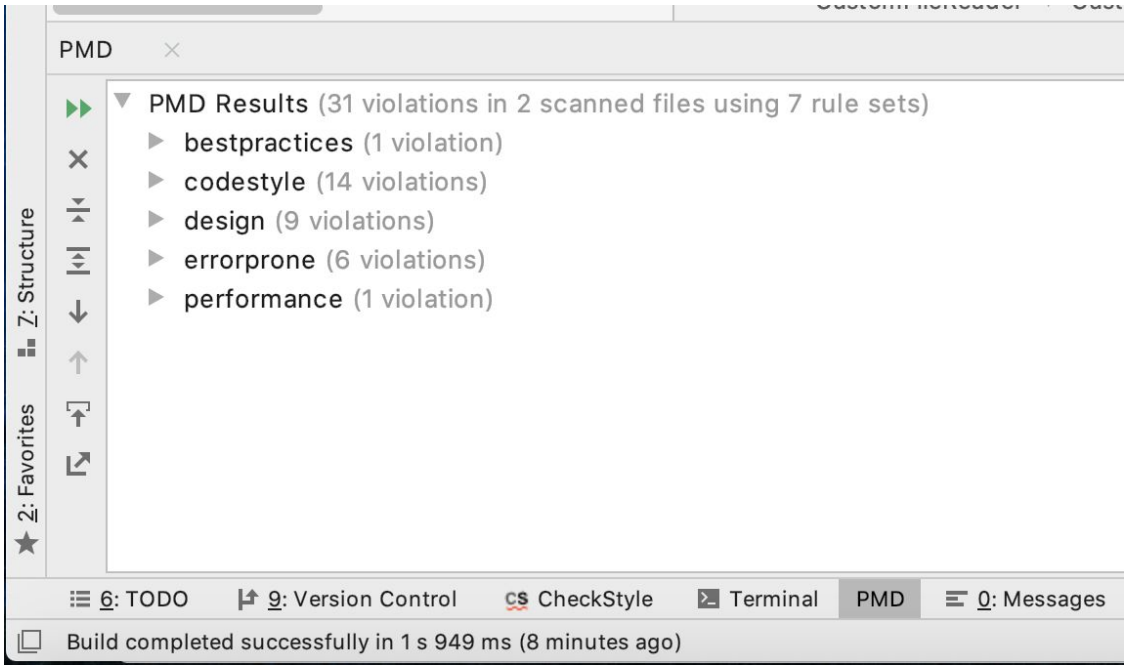
5. Something that is easy to forget when writing your own equals method is that a null object can be passed in as a parameter. Since we don’t do a null check before we start calling methods on the object, we run the risk of a null pointer exception.

6. To fix this, we need to check if the object is null, and if it is, return false. Add an if statement similar to the one below at the top of your equals method.

```
if(object == null)
{
    return false;
}
```

7. Run FindBugs again and you should see an “Internationalization” bug. Open that one and you will see that it says “Found reliance on default encoding in new CustomFileReader(String)...”. The problem here is that we are not specifying the encoding of the file we are reading, so Java will use the default encoding. To learn more about this issue, you can read this stack overflow post and the corresponding answers: https://stackoverflow.com/questions/696626/java-filereader-encoding-issue. We will ignore this issue for now. You will end up fixing it in the lab as a result of fixing a PMD bug.

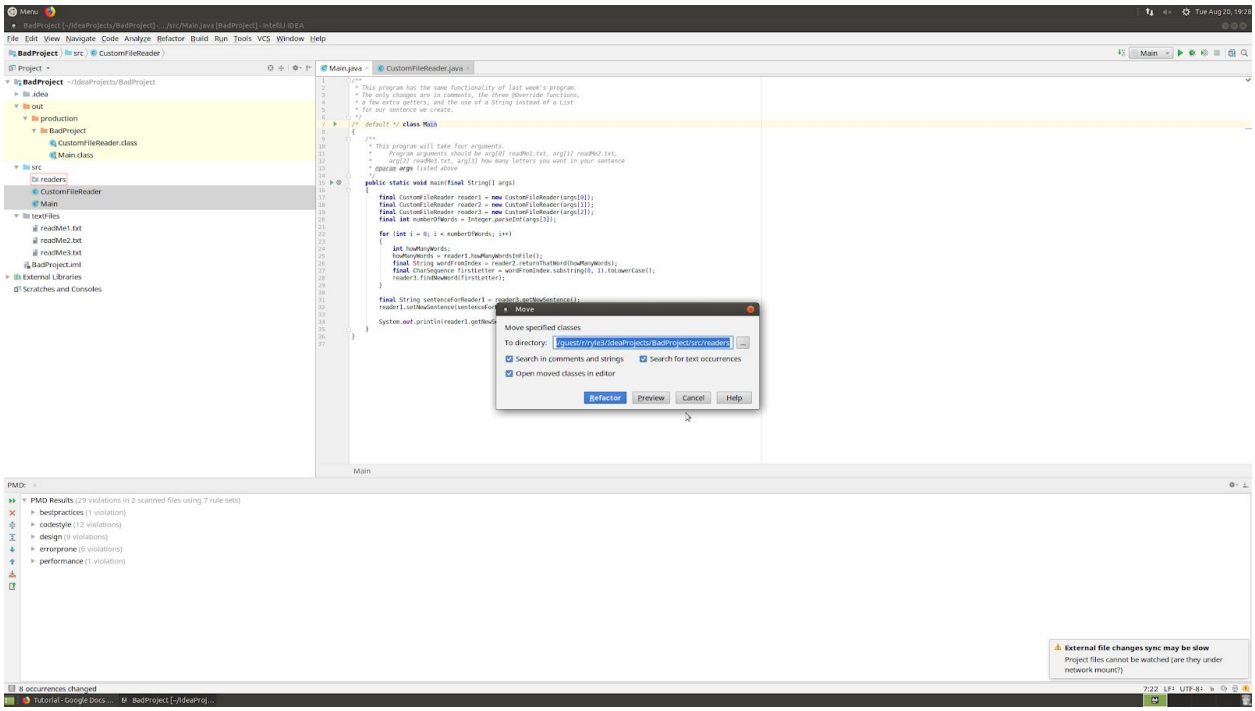
8. Now we will run PMD, and see if that static analyzer can help us improve our code as well. Right click on the src folder of your project and click Run PMD > Pre Defined > All. This is simply saying we want to run PMD using only the predefined rule set it comes with, and that we want the analyzer to check our code against all the predefined rules at once. What you should see in the window at the bottom is that there are 31 violations in 2 scanned files using 7 rule sets. When we first ran it there were many more, but we took care of the more mundane violations for you (hence all the comments and uses of the keyword final).



9. Just like last week, we will go through a few of these errors with you, but then you will fix the rest on your own. The first violation we will look at is under the topic “codestyle”.

Expand that list and look for the one that starts with the word “NoPackage”. PMD doesn’t like that we are using a default package for all of our java code.

10. To fix this, we will create a package and move our code into it. Right-click on the folder that contains your CustomFileReader and Main classes and select New > Package. Name the package “reader”. Now move Main and CustomFileReader into the package you just created. When you try to do that, a window will come up like the one in the image below. The problem is that by moving your classes into another package, the relative path of the project, your dictionary text files, and many other things in relation to your java code has now changed. It would be annoying to have to fix all of these things manually, but luckily it’s already built into IntelliJ. Click on the blue “Refactor” button and everything will be fixed correctly. If you rerun PMD, you will see you are now down to 29 violations.



11. The next violation we will look at is also under “codestyle”. Expand the list and find the violation that starts with the words “ShortClassName”. This violation is pretty simple but the fix is a little different than you might think.

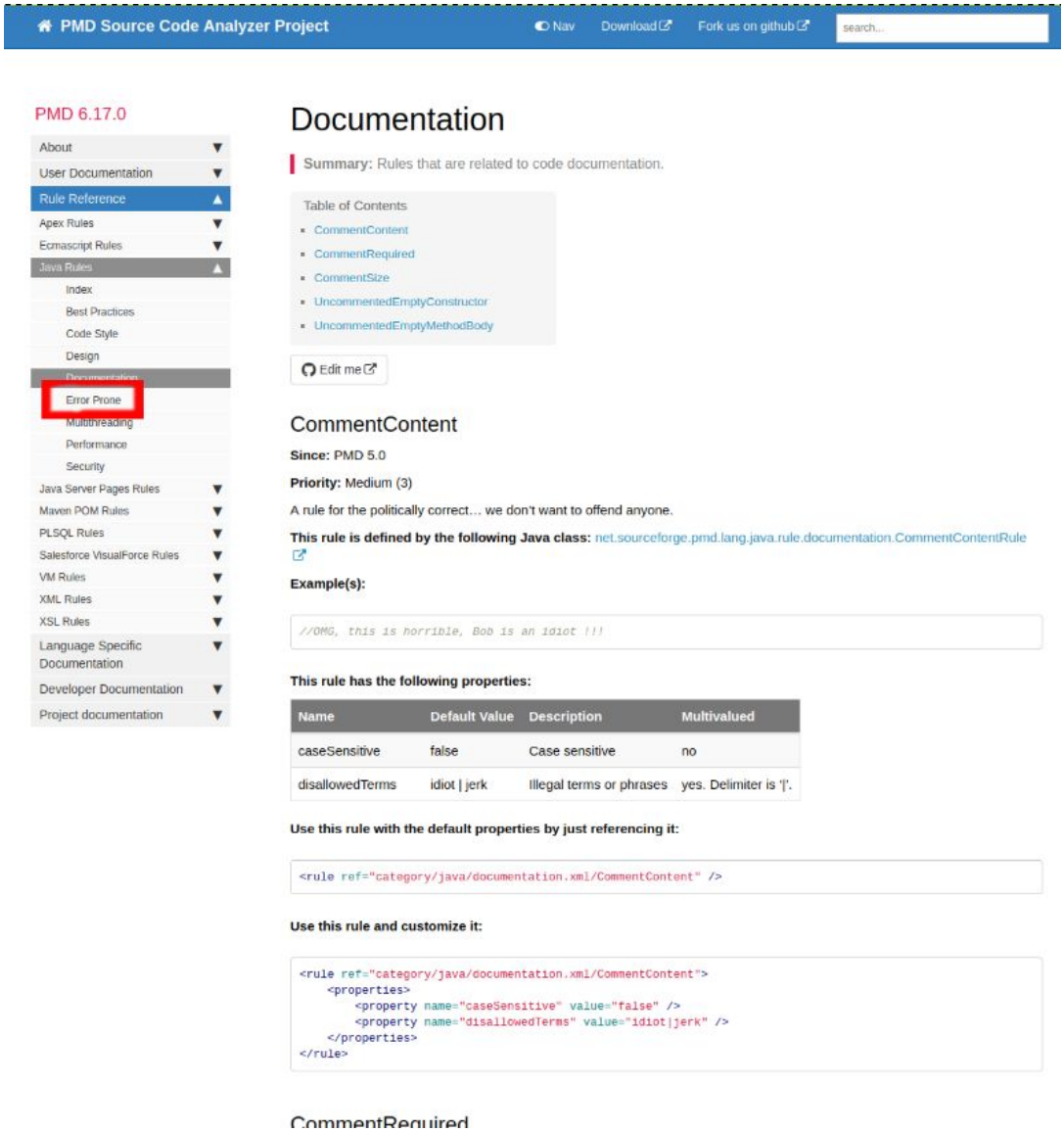
12. Let’s change the name of the class called “Main” to “Source”. Like before, if we plan on changing the structure of our project, we have to do a refactor. This is because our project currently knows our main method is in the class named “Main” not “Source”.

13. In the project window, right click on the Main class and then select Refactor > Rename… and type “Source”. Notice that this will also rename the class declaration inside of the file. That is because class declarations need to be named the exact same as the .java

file in which they are declared. If you simply typed “Source” where “Main” would have been in the class declaration inside of the .java file, another file would show up in your project window with that name. This would have resulted in an incorrect refactor which could cause problems. Remember, **always rename through refactoring**, never do it manually! Run PMD again, and you should be down to 28 violations.

14. The last violation we will fix together is under “errorprone”. Expand the list and look for the violation that starts with “UseLocaleWithCaseConversions”. This violation, like a few others you will encounter in the lab, doesn’t explain itself very well. We can use the public PMD guide to help us understand what this error means.

15. You will notice on this website, on the left, that “Java Rules” is expanded. Currently we are under the section that explains the different types of Documentation violations that PMD searches for but we want the “Error Prone” section. Click on that title (as seen below), and then click on “UseLocaleWithCaseConversion” from the alphabetized blue lettered list that shows up as a result.



16. If you read this section, you should be able to understand that PMD wants you to specify what language you are trying to use when calling .toLowerCase(). This is because other languages may have different characters than English when converting to lowercase or other things that would cause undefined behavior. We will specify the language we are using to be English by adding Locale.US inside the parameter of our lowercase function, like this .toLowerCase(Locale.US). This will ensure that if we call the lowercase function on any non English words, they will be ignored. Make sure to import the Locale data type in your Source class by moving your cursor prompt over the word, pressing Alt + Enter on your keyboard, and then clicking “import class”. Run PMD again, and you should be down to 27 violations

Lab

Now we will allow you to fix the remaining violations on your own. Your goal is to solve the following violations in PMD:

1) codestyle/ShortVariables
2) codestyle/LongVariables
3) performance/AvoidFileStream

Remember to use the PMD website as it can be a helpful hint in knowing what each violation means, and how to fix it.

Make sure that IntelliJ's static analyzer and FindBugs don't detect any further violations in your code while making your changes. Also, run the program and make sure it still works as before.

Submit all the items listed below in the "Submission" section to Canvas by the due date to receive credit for this assignment.

Submission

- Your revised Main.java and CustomFileReader.java classes,
- A screenshot of the green bar or popup from IntelliJ's static analyzer listing no suspicious code,
- A screenshot of the green box from FindBugs listing 0 bugs found in your 2 classes,
- A screenshot of your PMD violations being listed as either 24 or less.