Objectives:

- Use a code quality metric tool for preemptive defect detection.
- Determine Key Characteristics of Code Inspection Tools

Requirements

- Java 8 or newer JRE/JDK
- Eclipse IDE for Java Developers Download from Eclipse || Get executable
- Git

Introduction

PMD is a **source code analyzer**. It finds **common programming flaws** like unused variables, empty catch blocks, unnecessary object creation, and so forth [1]. Like other tools, PMD can **verify that coding conventions and standards are followed.** PMD is more focused on **preventive defect detection**. It comes with a vast set of rules and is highly configurable. PMD can also configure - in a simple way - particular rules to use in a specific project [2].

PMD integrates well with IDEs such as Eclipse and NetBeans, and it also fits well into the build process thanks to its smooth integration with Ant and Maven [2].

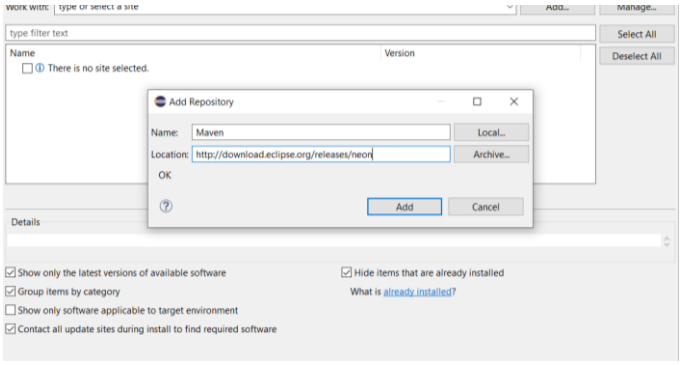
For this lab, we will use **Eclipse and Maven** together with **PMD** to inspect the source code of a project.

Informative only – How to add PMD in Java with Maven

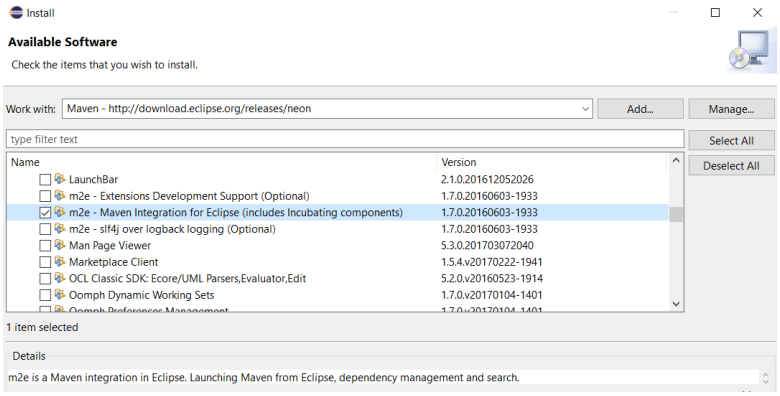
Part 1: Install Maven for Eclipse.

Most Eclipse downloads include the maven tooling already. If this is missing in your installation, follow these steps, else jump to part two:

1. Open the plugin installation window by selecting "**Help → Install new software**".
2. Click on Add and type "**Maven**" for the name and http://download.eclipse.org/releases/neon for the location.



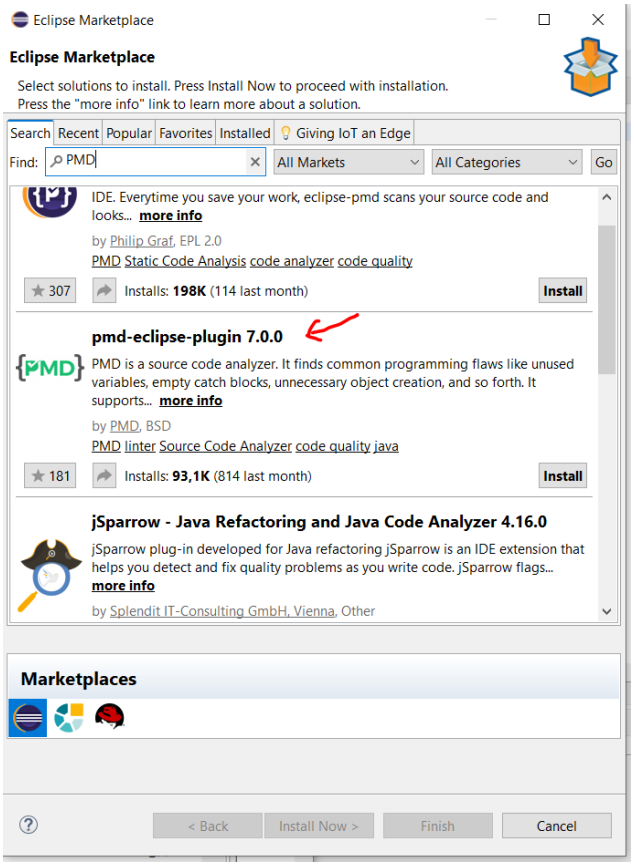
3. Click "**Add**" again and wait until the process finishes.
4. Search Maven and check "**Maven Integration for Eclipse**" under "**General Purpose Tools**"



5. Click "**Next**" and step through the following installation screens.

Part 2: Install the Eclipse PDM plug-in

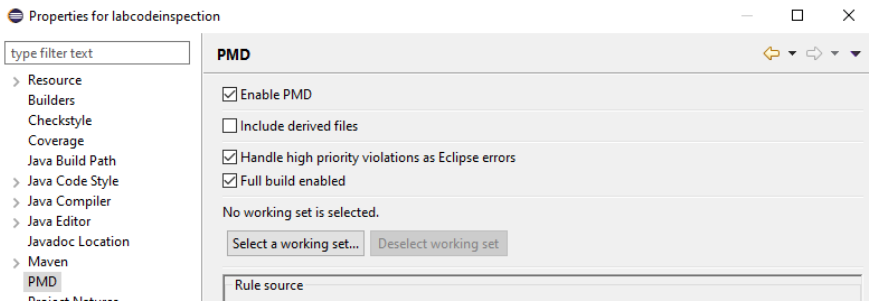
1. Open the plugin installation window by selecting the "**Help → Marketplace**".
2. Search for the PMD plugin
3. Install the selected Plugin



Part 3: Download and configure the project.

Open the project you will use in eclipse.
Make sure you use your own repository after cloning it.

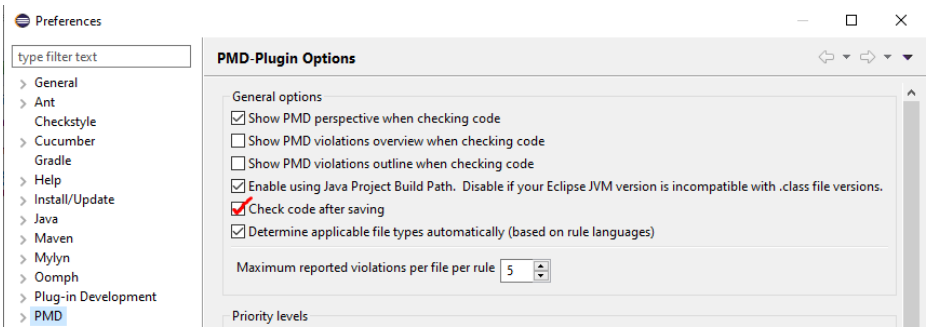
1. PMD will not be activated for the project by default. Open the project properties window by clicking in "**Project → Properties**".
2. Select "PMD" on the side bar and check "Enabled PMD ".



3. Look at all the rulesets that come with PMD, leave the default set of rules.
4. Click "**Apply and Close**" and "**Yes**".

Part 4: Using PMD

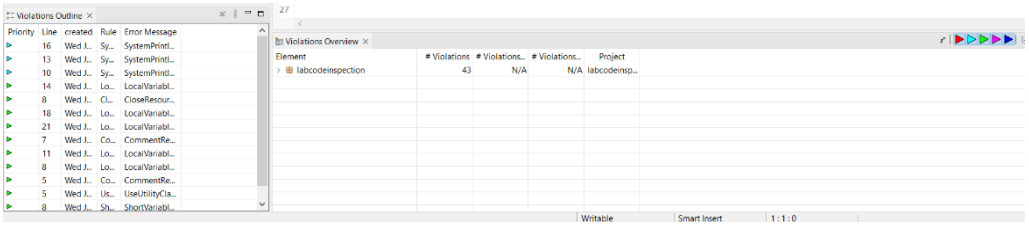
Go to "**Window → Preferences**", select "**PMD**" and check "**Check code after saving**"



To run PMD, right click on the project and select "**PMD → Check code**"

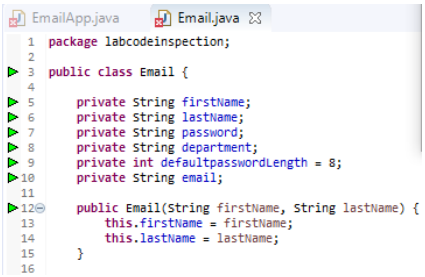
Two new windows are displayed with all violations. Each violation has its priority represented by a color and corresponding rule. The meaning of the colors is:

- **Red** is blocker → High priority.
- **Cyan** is critical → Medium priority.
- **Green** is urgent → Medium priority.
- **Pink** is Important → Medium priority.
- and **Blue** is Warning → Low priority.



If you see that files are duplicated in the Violation Overview Window, check code again.

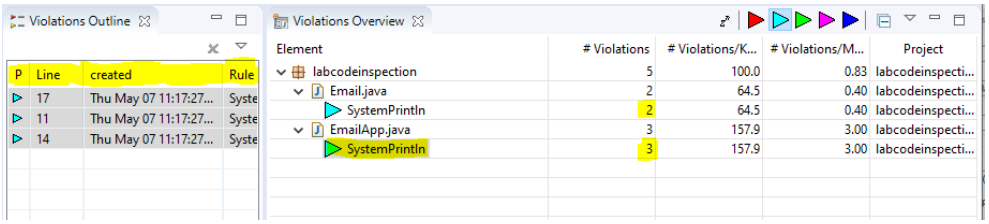
PMD shows the violations next to the lines that generate them.



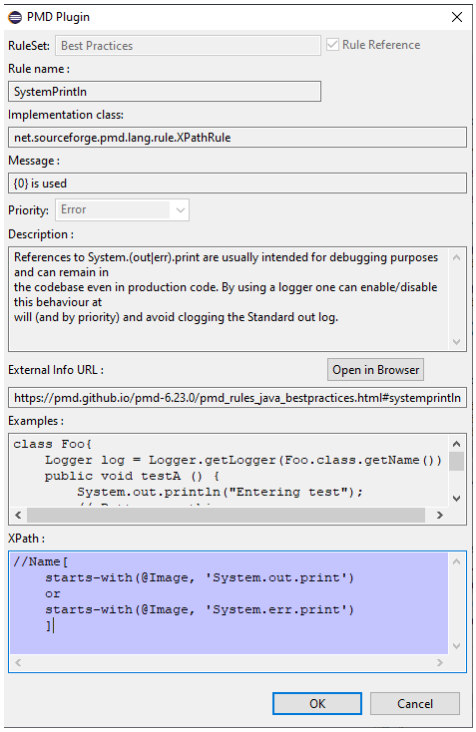
We can filter violations using the color indicators on the top right of Violations Overview window.

For example, if we filter only by critical violations, Violations Overview window shows that Email.java and EmalApp.java have 2 and 3 violations respectively for the rule "**SystemPrintln**".

If we double-click on an element, the Violation Outline window will update showing all errors related to a file and some important data such as the violation line, the affected rule, and the error message.



At "**Violations Outline**" tab, we can right-click on a violation and select "Show details..." for more information and an example solution.

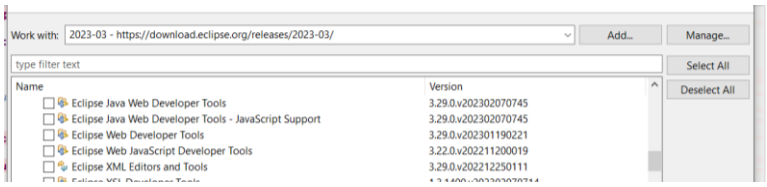


For more information about configurations, refer to section 2 and 3 from chapter 22 of the book [2] and PMD website [1].

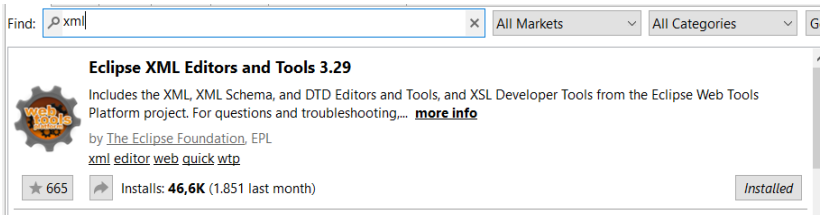
Part 5: Creating PMD rulesets

A PMD ruleset is simply an XML file that lists a set of rules that fit the project. You can include entire rulesets, or selectively choose specific rules from within other rulesets. You can also provide extra parameters to certain rules to customize their behavior. To do so, follow these steps:

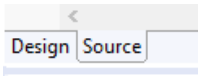
1. Right click on the project, then "**New → Other → XML → XML File**".
2. If the option does not appear, install it in "**help → Install new software**", with **Eclipse XML Editors and Tools** in the section **Web, XML, Java EE and OSGi Enterprise Development**.



And in the marketplace search for XML Editors and Tools



3. Select the project, enter a file name as "**<your name>_ ruleset**"
4. Click "Source" in the bottom tab to change the view and edit the file directly.



5. Here is a fragment of a typical configuration document, copy and paste into the file:

```
<?xml version="1.0" encoding="UTF-8"?>

<ruleset name="<your name> Rules"
   xmlns="http://pmd.sourceforge.net/ruleset/2.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://pmd.sourceforge.net/ruleset/2.0.0
https://pmd.sourceforge.io/ruleset_2_0_0.xsd">

  <description>
    Code Inspection Lab, <your full name>
  </description>

</ruleset>
```

6. Let's reference a complete ruleset. Add the following line below the description tag:

```
<rule ref="category/java/performance.xml" />
```

This ruleset comes by default with PMD and it has rules that flag suboptimal code.

7. Now, add another reference, but exclude some rules from the ruleset:

```
<rule ref="category/java/bestpractices.xml">
        <exclude name="SystemPrintln" />
</rule>
```

This ruleset also comes by default with PMD and it has rules which enforce generally accepted best practices but excluding "SystemPrintln" rule.

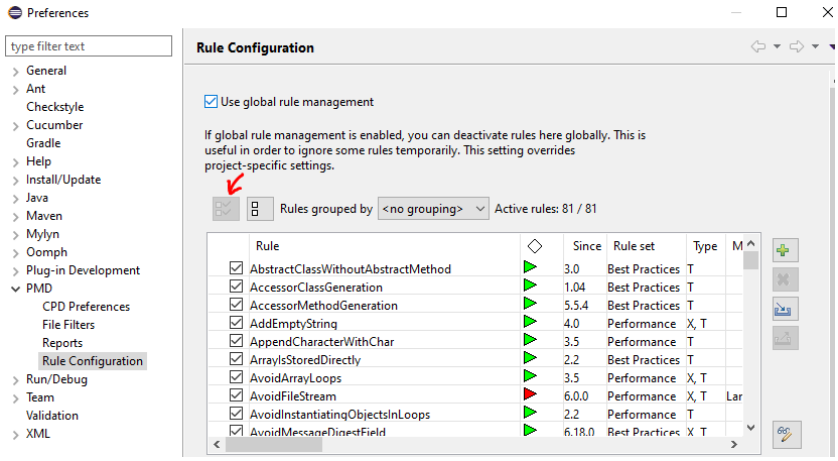
8. We can add rules from a specific ruleset as follow:

```
<rule ref="category/java/design.xml/ImmutableField" />
<rule ref="category/java/design.xml/UseUtilityClass">
        <priority>1</priority>
</rule>
```

Here, we are adding “**ImmutableField**” rule and “**UseUtilityClass**” rule from the Design ruleset and changing its priority to 1. Priority is an integer ranging from 1 to 5, with 1 being the highest priority.

For more information about PMD rulesets, refer to PMD documentation [3]

9. To use an external ruleset, we need to go to the PMD Configuration Window. Go to “**Window → Preferences → PMD → Rule Configuration**”.
10. Check “**Use global rule management**”, then group rules by “**Rule Set**”.
11. Select all the rule sets and click in the “X” button to delete them.
12. Click "**Import rule set...**" (under the "**x**"), browse your file and Click “**Ok**”.
13. The rules we added are not listed yet, press "**Apply and close**", then "**Yes**".
14. Return to PMD Configuration Window. Notice that the checkbox next to the rule names is unchecked.

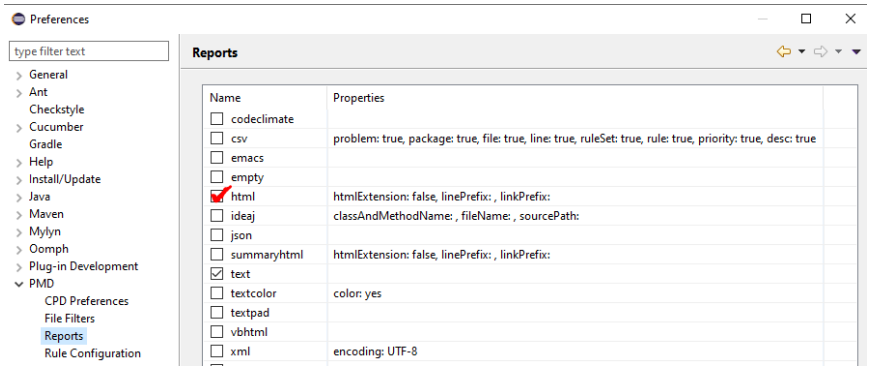


15. Press "**Apply and close**", then "**Yes**".
16. Right click on the project and select “**PMD → Clear Violations**”. Then “**PMD → Check Code.**”

Note that it is possible to configure the properties for each rule we add. To learn more about rulesets and their properties, refer to PMD Java Rules [3]

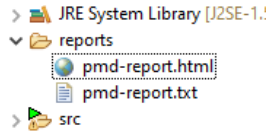
Part 6: Generating a PMD report

1. Open PMD configuration window, select Reports and check “html.”



2. Right-click the project then click “**PMD → Generate Report**.”

3. A folder named reports is created in the tree project. Open it and double click the "**pmd-report.html**" to see the full report.



Part 7: Suppressing PMD Rules

Sometimes you will have a legitimate reason for not respecting one of the PMD rules. PMD provides several methods by which Rule violations can be suppressed. We will be using comments and annotations.

1. Go to Email.java. See that line 5 has a violation related to ImmutableField rule.
2. Write "NOPMD" as a comment in the same line where the violation occurred.
3. Optionally, add a message placed after the NOPMD marker. This will get placed in the report.

```
3  public class Email {
4
5      private String firstName; //NOPMD This field will be manipulated later.
6      private String lastName;
7      private String password = null;
8      private String department;
9      private int defaultpasswordLength = 8;
10     private String email;
11
```

4. Go to EmailApp.java and check the violation. The rule violated is "**UseUtilityClass**".
5. Write an annotation above the line as follow:

```
5  @SuppressWarnings("PMD.UseUtilityClass")
6  public class EmailApp {
```

   Please note that only that rule will be ignored.

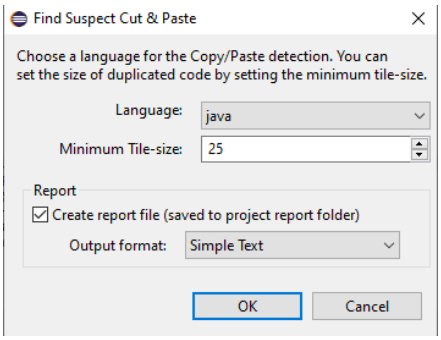
6. Save the file and see the results.

For more information about suppressing rules, check the PMD website [4] and section 7 from chapter 22 of the book [2]

Part 8: Detecting Cut-and-Paste with CPD

PMD comes with a useful tool for detecting cut-and-pasted code called CPD (Cut-and-Paste Detector). Follow these steps to use it and generate a report.

1. Just for demonstration purpose, copy and paste the "**randomPassword**" method from Email.java to EmalApp.java.
2. Right-click the project and select "**PMD → Find Suspect Cut and Paste**" from menu options.
3. Select "**java**" for Language, then click "**Ok**".



4. CPD View Window will open with the results. Also, a text file called cpd-report.txt will be generated in the /report directory.

```
Found a 11 line (74 tokens) duplication in the following files:
Starting at line 33 of C:\Users\josea\OneDrive\Documentos\EclipseProjects\labcodeinspection\src\main\java\labcodeinspection\Email.java
Starting at line 26 of C:\Users\josea\OneDrive\Documentos\EclipseProjects\labcodeinspection\src\main\java\labcodeinspection\EmailApp.java

    }

    private String randomPassword(int length) {
        String set = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890#$&@*";
        char[] password = new char[length];
        for (int i = 0; i < length; i++) {
            int rand = (int) (Math.random() * set.length());
            password[i] = set.charAt(rand);
        }
        return new String(password);
    }
```

5. Revert the changes made in EmailApp.java.

Part 9: PMD and Maven

1. Open “pom.xml” file

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.espol.edu</groupId>
    <artifactId>labcodeinspection</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Lab Code Inspection</name>
    <description>Software Engineering II</description>
</project>
```

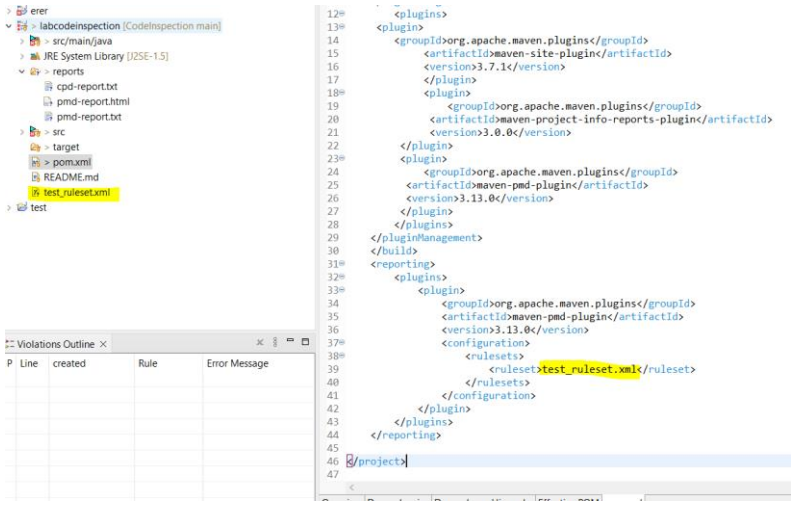
2. Add the following lines under project tag to install all the necessary plugins:

```
<build>
  <pluginManagement>
    <plugins>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-site-plugin</artifactId>
                <version>3.7.1</version>
          </plugin>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-project-info-reports-plugin</artifactId>
                <version>3.0.0</version>
          </plugin>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-pmd-plugin</artifactId>
                <version>3.13.0</version>
          </plugin>
    </plugins>
  </pluginManagement>
</build>
```

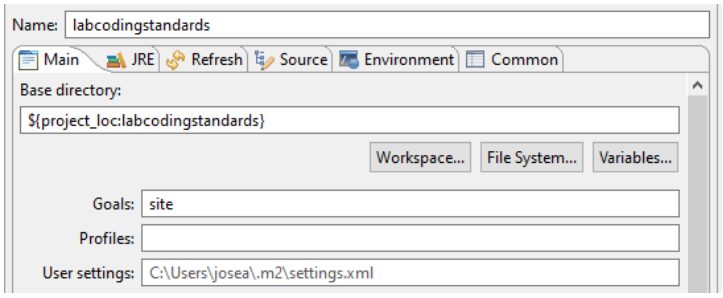
```
<reporting>
  <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-pmd-plugin</artifactId>
          <version>3.13.0</version>
          <configuration>
            <rulesets>
                  <ruleset><yourname>_ruleset.xml</ruleset>
                </rulesets>
          </configuration>
        </plugin>
  </plugins>
</reporting>
```

The ruleset tag is used to specify a file that contains rules to use in the checking process. In this case, we are telling the plugin to use our ruleset file.

Like this:



3. Save the file, right Click on it, then "**Run as → Maven build…**"
4. Type "**site**" for the Goals and Click Run.

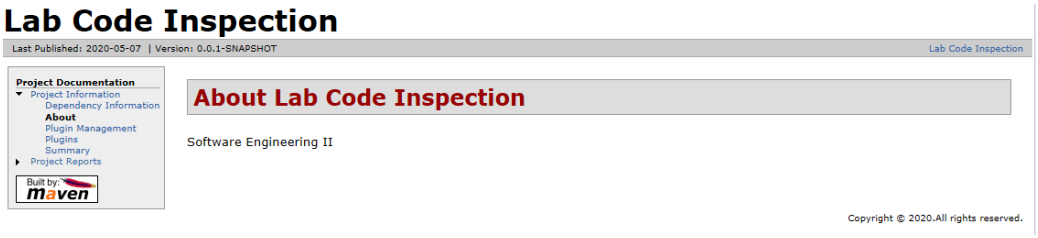


5. Wait for the process to finish.

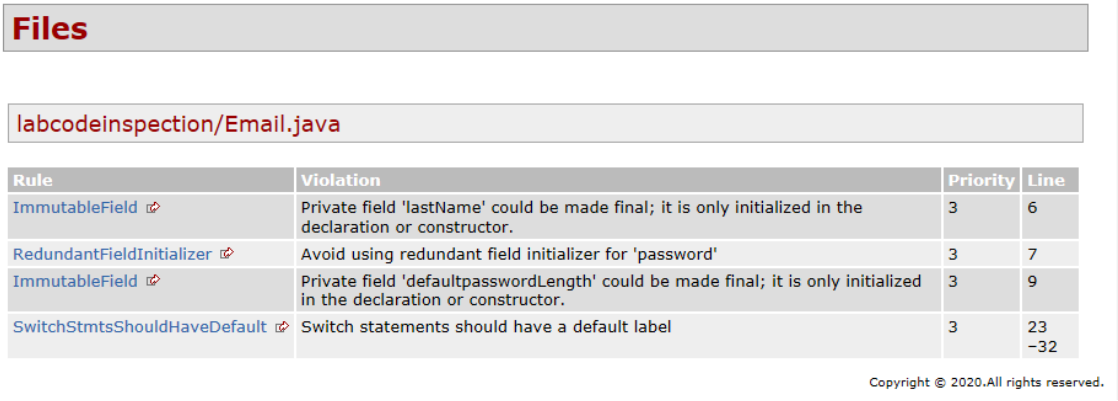
```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  17.972 s
[INFO] Finished at: 2020-05-07T23:17:24-05:00
[INFO] ------------------------------------------------------------------------
```

6. Go to your project directory, open "**target→ site**". This folder is visible from the tree project in Eclipse too.

7. Several html files are shown. Open “**index.html**".



8. Click on “**Project Reports → PMD**” to see a detailed PMD report of your project.



Development

1. Create the code for the requirements given at the end of this document.
2. Follow the steps in the instructive manual and add the code inspection tool to your project.
3. Generate a first report result.
4. Correct any flag that is given in the first report.
5. Generate a second report with the corrected errors.

Challenge

- Create a branch in the repository of the previous workshop, CODING STANDARDS - I TERM 2023, and add the new requirements for the vacation system:

  For this new version of the system, the client has added some requirements that will increment the user experience.

  - The system should allow the user to select optional add-ons to customize their vacation package. The available add-ons are:
    - All-Inclusive Package - $200 per traveler
    - Adventure Activities Package - $150 per traveler
    - Spa and Wellness Package - $100 per traveler
  - The system should calculate the cost of the selected add-ons and include it in the total cost of the vacation package.

- Configure PMD for that project and generate its respective report.
- Create your **own configuration file** for this case and add the rules you think are necessary, minimum 4.
- Correct any violation in the code that have been generated in the report.
- Generate a new report (without violations).

Deliverables

1. Practice report with at least: cover, introduction, development, conclusions and recommendations, and references.
2. Two PMD reports.
3. Challenge
4. Include in the report the **url** of the repository where you performed the lab.

Rubric

Description	Value
Lab report Criteria: lab report content (30 pts) • All the evidences: screenshots of the process, initial and fixed reports, challenge and url are included: 30 pts • Just both reports are included: 15 pts • Empty, partial or undelivered report: 0 pts Criteria: fixed issues (20 pts) • All the violations were fixed: 20 pts • Some violations were fixed: 5 pts • Empty report or neither violation fixed: 0 pts Criteria: support (10 pts) • The fixed PMD report should be in concordance with the source-code (repository): 10 pts • Otherwise: 0 pts	60
Challenge Criteria: Code generation (20 pts) • The branch fulfills the new requirements of the system (20 pts) • There is not addition of the new requirements (0 pts) Criteria: PMD tool (20 pts) • There is evidence of the PMD tool use, and the fixes are corrected (20 pts) • There is no report (0 pts)	40
Penalty for not having **url of a public repository**	-100

Late Submission Policy

Delay (§)	Penalty (Ω)
1 hour or less	loss of 10%
1 to 6 hours	loss of 20%
6 to 24 hours	loss of 30%
Over 24 hours:	loss of 100%

(§) every clock hour counts including weekends or holidays.
(Ω) automatic and non-negotiable penalty

References

[1] PMD, "PMD Source Code Analyzer," [Online]. Available: https://pmd.github.io/.

[2] J. Ferguson, Java Power Tools, O'Reilly Media, 2008.

[3] PMD, "Java Rules," [Online]. Available: https://pmd.github.io/latest/pmd_rules_java.html.

[4] PMD, "Suppressing warnings," [Online]. Available: https://pmd.github.io/latest/pmd_userdocs_suppressing_warnings.html.

[5] Instructions, https://github.com/leortyz/softwareEngineeringResources/wiki/Code-Inspection

Dining Experience Manager

Description: Create a command-line application for managing dining experiences and calculating the total cost based on the selected meals and quantities.

Requirements:

1. **Menu and Meal Selection**
   - The system should display a menu with various dining options and their corresponding prices.
   - Users can select multiple meals to order and specify the quantity for each meal.
2. **Meal Quantity Validation**
   - The system should validate that the quantity entered for each meal is a positive integer greater than zero.
   - If invalid quantities are entered, the Dining Experience Manager should prompt users to re-enter the quantities.
3. **Cost Calculation**
   - The base cost for a dining experience is $5, and it should be applied to every order.
   - If the total quantity of meals ordered is more than 5, apply a discount of 10% to the total cost.
   - If the total quantity of meals ordered is more than 10, apply a discount of 20% to the total cost.
4. **Special Offer Discount**
   - The system should have the ability to apply special offer discounts based on certain conditions.
   - If the total cost of the meal order exceeds $50, apply a discount of $10 to the total cost.
   - If the total cost of the meal order exceeds $100, apply a discount of $25 to the total cost.
5. **Meal Availability**
   - The system should validate that the selected meals are available on the menu.
   - If an unavailable meal is selected, the Dining Experience Manager should display an error message and prompt users to re-select the meals.
6. **Maximum Order Quantity**
   - The system should handle orders with a maximum quantity of 100 meals.
   - If users attempt to order more than 100 meals, the Dining Experience Manager should display an error message and prompt them to re-enter the quantities.

7. **User Confirmation**
   - Before finalizing the order, the system should display the selected meals, their quantities, and the total cost for user confirmation.
   - Users can either confirm the order or cancel and make changes to the selections.
8. **Output**
   - Provide the total cost of the dining experience as a positive integer if the order is confirmed and valid.
   - Return a value of -1 if the input data is not valid or if the order is canceled.
9. **Error Handling**
   - The system should handle errors gracefully and provide clear error messages if users enter invalid input or if there are any calculation errors.

Note: The -1 output should only be considered as a last resort if the input data is invalid, so the system is expected to validate every input the user enters.