PMD Tool – Installation and Usage Guide

Summary: This documentation contains the steps for installation and usage of PMD plugin (on Eclipse IDE). Main purpose of this document is to provide a kickstart with the tool and readers may consider using more references to further deep dive into the topic.

Contents

About the Tool

- **PMD** is a static source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It's mainly concerned with **Java and Apex** but **supports 12 other languages**.

  PMD features many **built-in checks** (in PMD lingo, *rules*), which are documented for each language in their Rule references. PMD also supports an extensive API to **write your own rules**, which you can do either in Java or as a self-contained XPath query.

  PMD is most useful when **integrated into your build process**. It can then be used as a quality gate, to enforce a coding standard for your codebase. Among other things, PMD can be run:

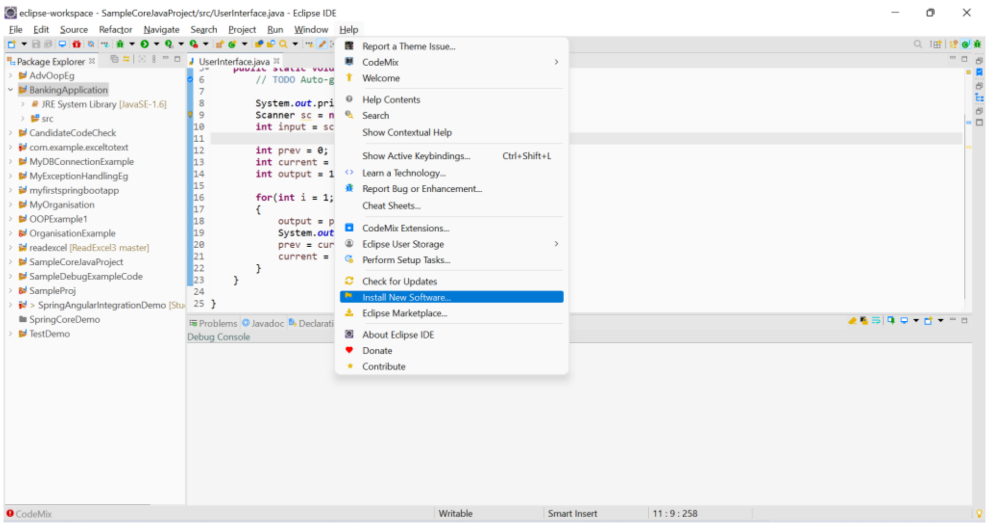
  - As a Maven goal
  - As an Ant task
  - As a Gradle task
  - From command-line

- **CPD**, the **copy-paste detector**, is also distributed with PMD. You can also use it in a variety of ways, which are documented here.

Installing PMD plugin on Eclipse

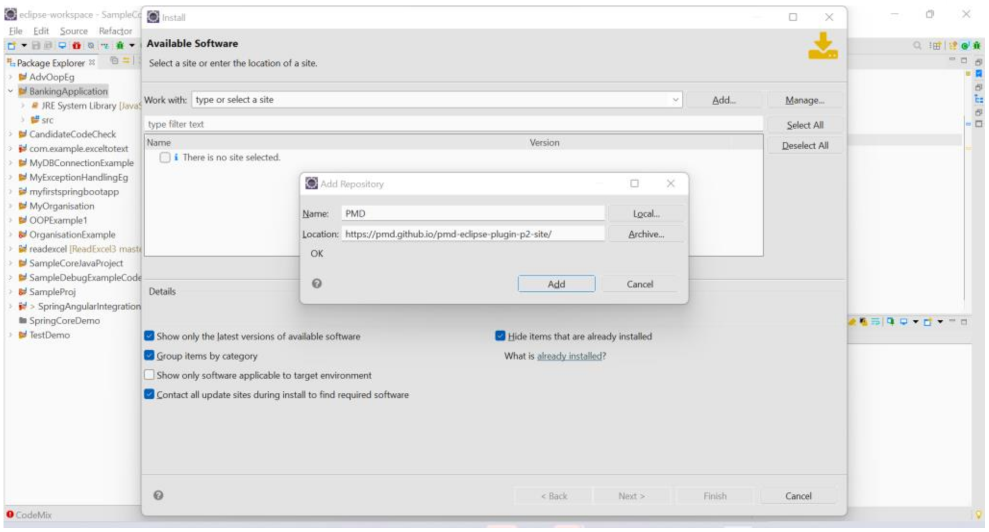
Steps to install the PMD plugin for Eclipse:

- Start Eclipse and open a project
- Select "Help"->"Install New Software"->" Install" Dialog Box will open



```
    public static void
        // TODO Auto-g

        System.out.pri
        Scanner sc = n
        int input = sc

        int prev = 0;
        int current =
        int output = 1

        for(int i = 1;
        {
            output = p
            System.out
            prev = cur
            current =
        }
    }

}
```

- Click on "Add…" button in front of "Work with: "


Available Software

Select a site or enter the location of a site.

Work with: type or select a site

type filter text

There is no site selected.

Add Repository

Name: PMD

Location: https://pmd.github.io/pmd-eclipse-plugin-p2-site/

OK

[✓] Show only the latest versions of available software

[✓] Group items by category

[ ] Show only software applicable to target environment

[✓] Contact all update sites during install to find required software

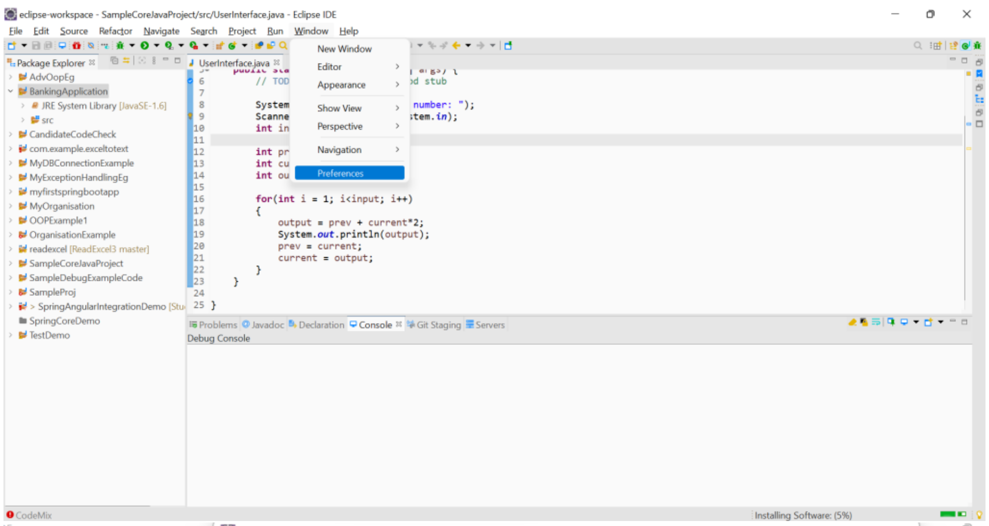
[✓] Hide items that are already installed

What is already installed?

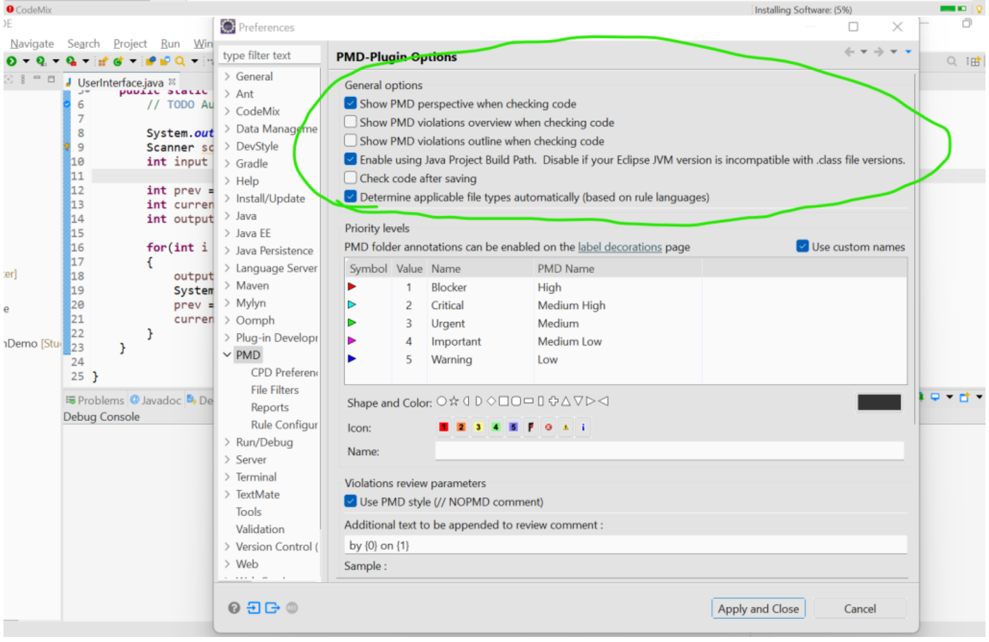
- Enter “PMD” into the Name field and https://pmd.github.io/pmd-eclipse-plugin-p2-site/ into the URL field
- Click through the rest of the dialog boxes to install the plugin.
- Accept the terms of license agreement on last screen and click on “Finish” button.

(Alternatively, you can download the latest zip file and follow the above procedures except for using “New local site” and browsing to the downloaded zip file.)

To configure PMD, select “Windows”->”Preferences”, then select PMD


eclipse-workspace - SampleCoreJavaProject/src/UserInterface.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
New Window
Editor
Appearance
Show View
Perspective
Navigation
Preferences
Problems Javadoc Declaration Console Git Staging Servers
Debug Console

```
        for(int i = 1; i<input; i++)
        {
            output = prev + current*2;
            System.out.println(output);
            prev = current;
            current = output;
        }
    }
}
```



PMD-Plugin Options

General options

- [✓] Show PMD perspective when checking code
- [ ] Show PMD violations overview when checking code
- [ ] Show PMD violations outline when checking code
- [✓] Enable using Java Project Build Path. Disable if your Eclipse JVM version is incompatible with .class file versions.
- [ ] Check code after saving
- [✓] Determine applicable file types automatically (based on rule languages)

Priority levels

PMD folder annotations can be enabled on the label decorations page — [✓] Use custom names

Symbol	Value	Name	PMD Name
▶	1	Blocker	High
▷	2	Critical	Medium High
▶	3	Urgent	Medium
▶	4	Important	Medium Low
▶	5	Warning	Low

Shape and Color:

Icon:

Name:

Violations review parameters

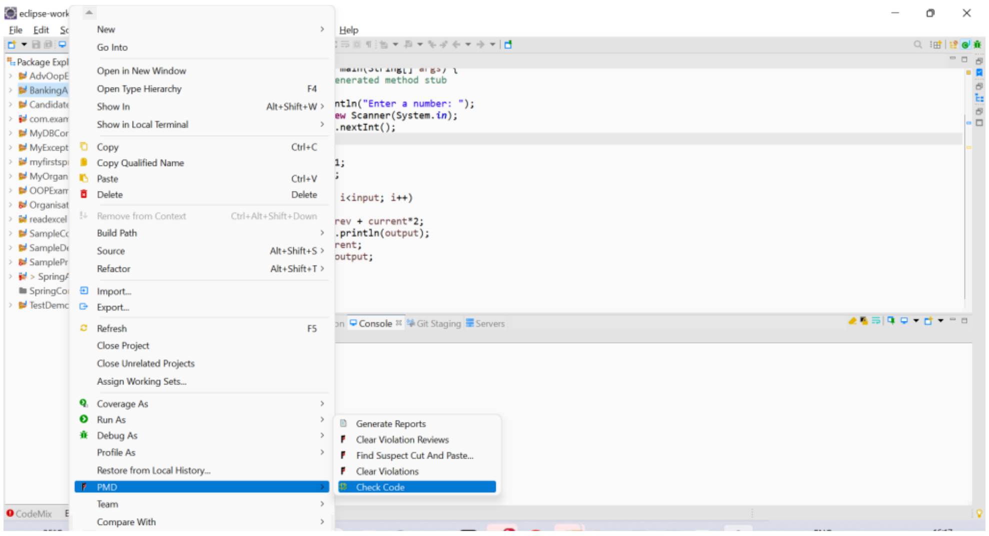
- [✓] Use PMD style (// NOPMD comment)

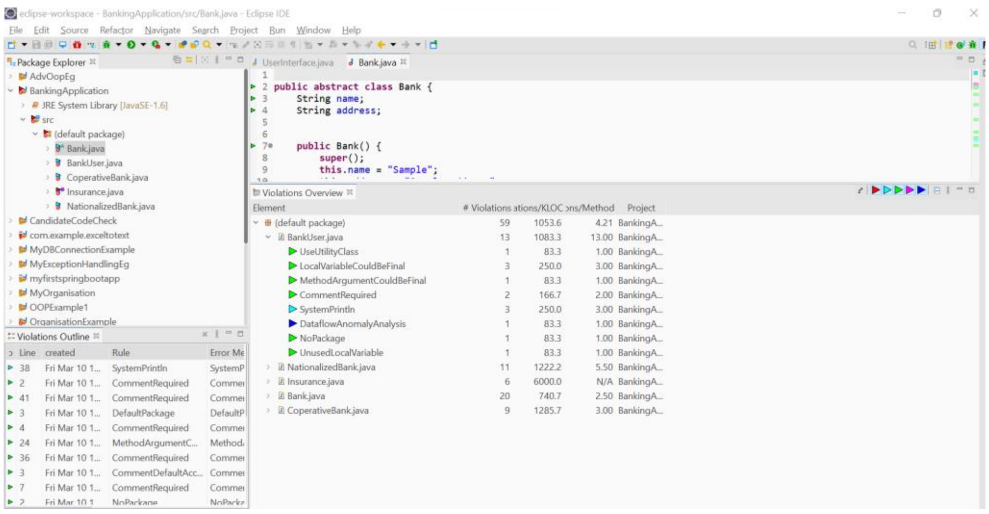
Additional text to be appended to review comment :

by {0} on {1}

Sample :

To run PMD, right-click on a project node and select “PMD”->”Check code with PMD”





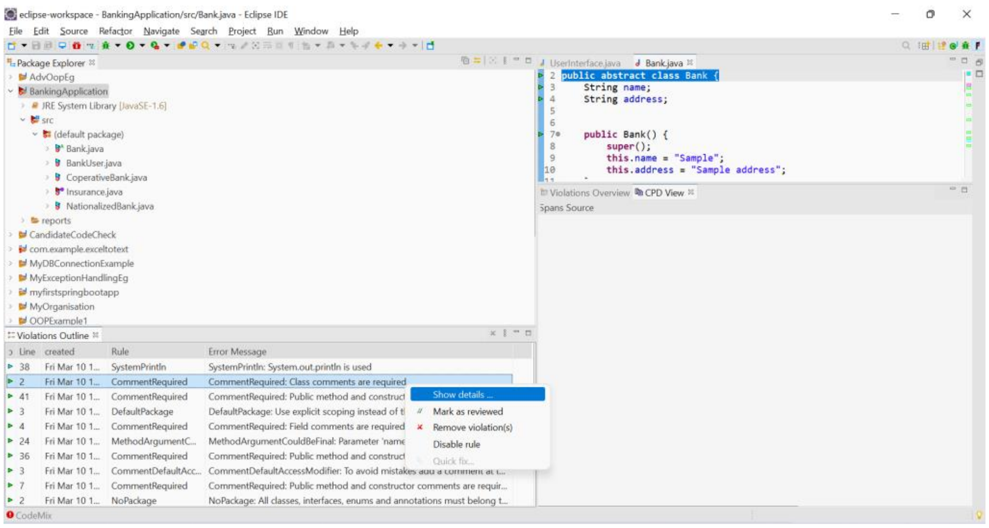
```
public abstract class Bank {
    String name;
    String address;


    public Bank() {
        super();
        this.name = "Sample";
```

Violations Overview

Element	# Violations	ations/KLOC	ɔns/Method	Project
(default package)	59	1053.6	4.21	BankingA...
BankUser.java	13	1083.3	13.00	BankingA...
UseUtilityClass	1	83.3	1.00	BankingA...
LocalVariableCouldBeFinal	3	250.0	3.00	BankingA...
MethodArgumentCouldBeFinal	1	83.3	1.00	BankingA...
CommentRequired	2	166.7	2.00	BankingA...
SystemPrintln	3	250.0	3.00	BankingA...
DataflowAnomalyAnalysis	1	83.3	1.00	BankingA...
NoPackage	1	83.3	1.00	BankingA...
UnusedLocalVariable	1	83.3	1.00	BankingA...
NationalizedBank.java	11	1222.2	5.50	BankingA...
Insurance.java	6	6000.0	N/A	BankingA...
Bank.java	20	740.7	2.50	BankingA...
CoperativeBank.java	9	1285.7	3.00	BankingA...

Violations Outline

Line	created	Rule	Error Me
38	Fri Mar 10 1...	SystemPrintln	SystemP
2	Fri Mar 10 1...	CommentRequired	Commer
41	Fri Mar 10 1...	CommentRequired	Commer
3	Fri Mar 10 1...	DefaultPackage	DefaultP
4	Fri Mar 10 1...	CommentRequired	Commer
24	Fri Mar 10 1...	MethodArgumentC...	Method
36	Fri Mar 10 1...	CommentRequired	Commer
3	Fri Mar 10 1...	CommentDefaultAcc...	Commer
7	Fri Mar 10 1...	CommentRequired	Commer
2	Fri Mar 10 1	NoPackage	NoPacka

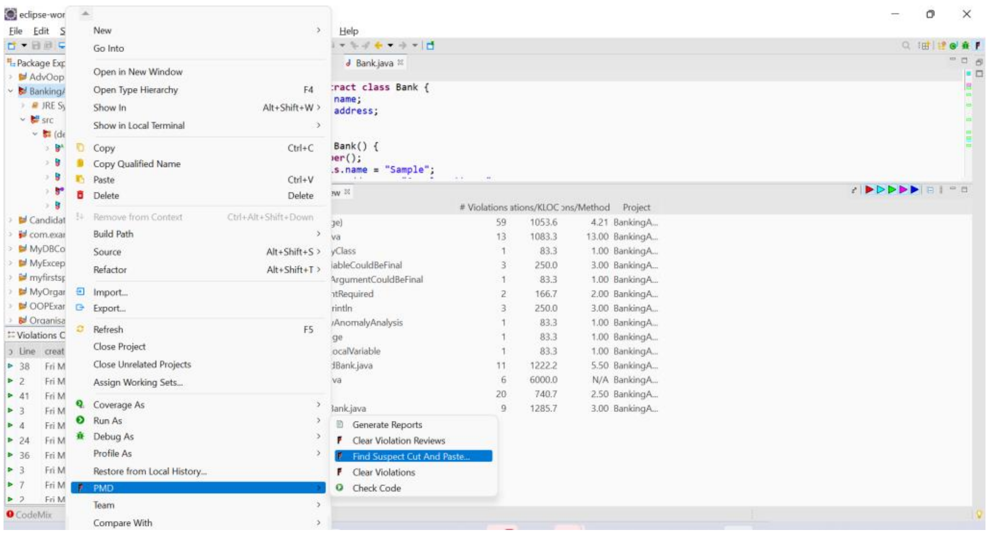


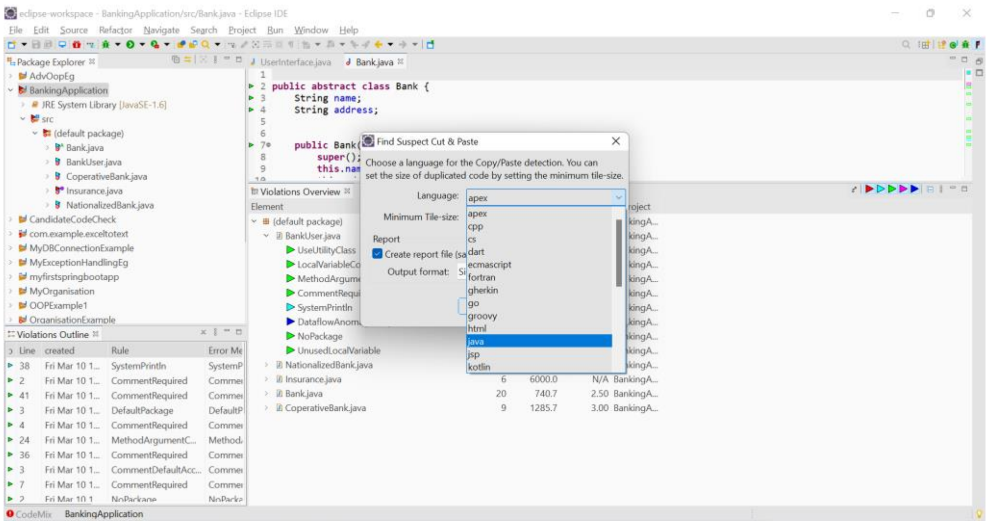
```
public abstract class Bank {
    String name;
    String address;


    public Bank() {
        super();
        this.name = "Sample";
        this.address = "Sample address";
```

Violations Outline

Line	created	Rule	Error Message
38	Fri Mar 10 1...	SystemPrintln	SystemPrintln: System.out.println is used
2	Fri Mar 10 1...	CommentRequired	CommentRequired: Class comments are required
41	Fri Mar 10 1...	CommentRequired	CommentRequired: Public method and construct
3	Fri Mar 10 1...	DefaultPackage	DefaultPackage: Use explicit scoping instead of t
4	Fri Mar 10 1...	CommentRequired	CommentRequired: Field comments are required
24	Fri Mar 10 1...	MethodArgumentC...	MethodArgumentCouldBeFinal: Parameter 'name
36	Fri Mar 10 1...	CommentRequired	CommentRequired: Public method and construct
3	Fri Mar 10 1...	CommentDefaultAcc...	CommentDefaultAccessModifier: To avoid mistakes add a comment at t...
7	Fri Mar 10 1...	CommentRequired	CommentRequired: Public method and constructor comments are requir...
2	Fri Mar 10 1...	NoPackage	NoPackage: All classes, interfaces, enums and annotations must belong t...

To run the duplicate code detector, right-click on a project node and select “PMD”->”Find suspect cut and paste”. The report will be placed in a “reports” directory in a file called “cpd-report.txt”





eclipse-workspace - BankingApplication/src/Bank.java - Eclipse IDE

```java

public abstract class Bank {
    String name;
    String address;


    public Bank(
        super();
        this.nam
```

Find Suspect Cut & Paste

Choose a language for the Copy/Paste detection. You can set the size of duplicated code by setting the minimum tile-size.

Language: apex

Minimum Tile-size:

Report

☑ Create report file (sa

Output format: Si

apex, cpp, cs, dart, ecmascript, fortran, gherkin, go, groovy, html, java, jsp, kotlin

Violations Outline

Line	created	Rule	Error Me
38	Fri Mar 10 1...	SystemPrintln	SystemP
2	Fri Mar 10 1...	CommentRequired	Commei
41	Fri Mar 10 1...	CommentRequired	Commei
3	Fri Mar 10 1...	DefaultPackage	DefaultP
4	Fri Mar 10 1...	CommentRequired	Commei
24	Fri Mar 10 1...	MethodArgumentC...	Method.
36	Fri Mar 10 1...	CommentRequired	Commei
3	Fri Mar 10 1...	CommentDefaultAcc...	Commei
7	Fri Mar 10 1...	CommentRequired	Commei
2	Fri Mar 10 1	NoPackage	NoPacka

Violations Overview

Element				roject
(default package)				kingA...
BankUser.java				kingA...
UseUtilityClass				kingA...
LocalVariableCo				kingA...
MethodArgume				kingA...
CommentRequi				kingA...
SystemPrintln				kingA...
DataflowAnom				kingA...
NoPackage				kingA...
UnusedLocalVariable				kingA...
NationalizedBank.java				kingA...
Insurance.java	6	6000.0	N/A	BankingA...
Bank.java	20	740.7	2.50	BankingA...
CoperativeBank.java	9	1285.7	3.00	BankingA...

To find additional help for other features, please read included help by selecting Help->Help Contents and browse the “How to…” section in the “PMD Plugin Documentation” book.

Simple Troubleshooting Tips

After installing an update, if you get an Exception such as “java.lang.RuntimeException: Couldn’t find that class xxxxx”, try deleting the ruleset.xml file in the .metadata/plugins/net.sourceforge.pmd.eclipse directory in your workspace.

To get Eclipse to not flag the @SuppressWarnings(“PMD”) annotation, look under the menu headings Java -> Compiler -> Errors/Warnings -> Annotations -> Unhandled Warning Token.

PMD Rulesets and Rules

Summary: A ruleset is an XML configuration file, which describes a collection of rules to be executed in a PMD run. PMD includes built-in rulesets to run quick analyses with a default configuration, but users are encouraged to make their own rulesets from the start, because they allow for so much configurability. PMD allows creation of your own rulesets.
Creation of such ruleset and multiple configuration features offered by rulesets is not under the scope of this documentation. If needed, same can be referred through the following links:
https://pmd.github.io/latest/pmd_userdocs_making_rulesets.html
https://pmd.github.io/latest/pmd_userdocs_configuring_rules.html

PMD Rules (both pre-defined and user-defined) fall under different Priority levels.

PMD violations are assigned a priority from 1 (most severe) to 5 (least severe) according to the rule's priority.

Rule priority may, of course, changes a lot depending on the context of the project. However, you can use the following guidelines to assert the legitimate priority of your rule:

1. **Change absolutely required.** Behaviour is critically broken/buggy.
2. **Change highly recommended.** Behaviour is quite likely to be broken/buggy.
3. **Change recommended.** Behaviour is confusing, perhaps buggy, and/or against standards/best practices.
4. **Change optional.** Behaviour is not likely to be buggy, but more just flies in the face of standards/style/good taste.
5. **Change highly optional.** Nice to have, such as a consistent naming policy for package/class/fields…

Best Practices

Choose the rules that are right for you

Running every existing rule will result in a huge number of rule violations, most of which will be unimportant. Having to sort through a thousand-line report to find the few you're really interested in takes all the fun out of things.

Instead, start with some selected specific rules, e.g. the rules that detect unused code from the category Best Practices and fix any unused locals and fields.

Then, run rules, that detect empty if statements and such-like. You can find these rules in the category Error Prone.

After that, look at all the categories and select the rules, that are useful for your project. You can find an overview of the rules on the Rule Index.

Use the rules you like via a custom ruleset.

PMD rules are not set in stone

Developers are free to pick the PMD rules which they like and to ignore or suppress the warnings which they don't like (or don't need for their project).

How can I Suppress PMD warnings?

Summary: Learn how to suppress some rule violations, from the source code using annotations or comments, or globally from the ruleset

a) Suppress warnings using Annotations

Though there are many ways to suppress the warnings, annotations could be one of the simple choices.

When using Java 1.5 or later, you can use annotations to work around your particular issue on a case-by-case basis for suppressing PMD warnings, like this:

Example 1:

```
// This will suppress all the PMD warnings in this class

@SuppressWarnings("PMD")

public class Bar {

   void bar() {

      int foo;

   }

}
```

Example 2: Alternatively, you can suppress one rule with an annotation like this:

```
// This will suppress UnusedLocalVariable warnings in this class

@SuppressWarnings("PMD.UnusedLocalVariable")

public class Bar {

   void bar() {

      int foo;

   }

}
```

Example 3: Multiple rules can be suppressed by providing multiple values, i.e.

```
@SuppressWarnings({"PMD.UnusedLocalVariable", "PMD.UnusedPrivateMethod"})
```

b) Suppress warnings using NOPMD Comment

Alternatively, you can tell PMD to ignore a specific line by using the "NOPMD" marker in a comment, like this:

```
public class Bar {
   // 'bar' is accessed by a native method, so we want to suppress warnings for it
   private int bar; //NOPMD
}
```

You can use whatever text string you want to suppress warnings, by using the -suppressmarker CLI option. For example, here's how to use TURN_OFF_WARNINGS as the suppressor:

```
$ cat Foo.java
public class Foo {
    void bar() {
        int x = 2; // TURN_OFF_WARNINGS
    }
}

$ ./run.sh pmd -d Foo.java -f text -R java-unusedcode -suppressmarker TURN_OFF_WARNINGS
No problems found!
UnusedLocalVariable rule violation suppressed by //NOPMD in /home/tom/pmd/pmd/bin/Foo.java
```

Note that PMD expects the //NOPMD marker to be on the same line as the violation. So, for example, if you want to suppress an "empty if statement" warning, you'll need to place it on the line containing the if keyword, e.g.:

```
$ cat ~/tmp/Foo.java
public class Foo {
    void bar() {
        int x = 42;
        if (x > 5) { // NOPMD
        }
    }
}
$ java net.sourceforge.pmd.PMD -d ~/tmp/Foo.java -f text -R java-basic
No problems found!
```

A message placed after the NOPMD marker will get placed in the report, e.g.:

```
public class Foo {
    void bar() {
        try {
            bar();
        } catch (FileNotFoundException e) {} // NOPMD - this surely will never happen
    }
}
```

PMD Ruleset file and Usage Guidelines

Please follow the below mentioned steps to configure Finishing School defined PMD rulesets in your project.

Prerequisite: PMD Plugin must be installed on your IDE (Eclipse/IntelliJ/STS)

1. Right click on your project to open the Project "Properties"
2. Select the "PMD" option from the left menu
3. Check the Enable PMD checkbox. "Rule Source" section will be enabled
4. Browse the file – "**FinishingSchoolPMDRules.xml**"
5. Click "Apply and Close"

- To check your project code against the defined rulesets –
  Right-Click the project and select the PMD > Check Code

- To Clear violations –

Right-Click the project and select the PMD > Clear Violations