



How you could tune your code quality in the development process

Artem Sokovets

February 2024

Agenda

1. About me
2. Context
3. Three brothers
4. CheckStyle Customization
5. PMD Customization
6. Methods for Customizing SonarQube Rules
7. Our experience (EK Payments)
8. Conclusion
9. QA

About me



[linkedin.com/in/artem-sokovets-24a9845b/](https://www.linkedin.com/in/artem-sokovets-24a9845b/)

Artem Sokovets

Ex.Engineering Manager/Tech lead in Sber
Senior Backend Developer in Emirates (Payments)
14+ years in IT

Java/Kotlin/Dart
SpringBoot/Android/FlutterSDK

System design
Hackathons
Writer
Courses
Startup experience

Context - No problem

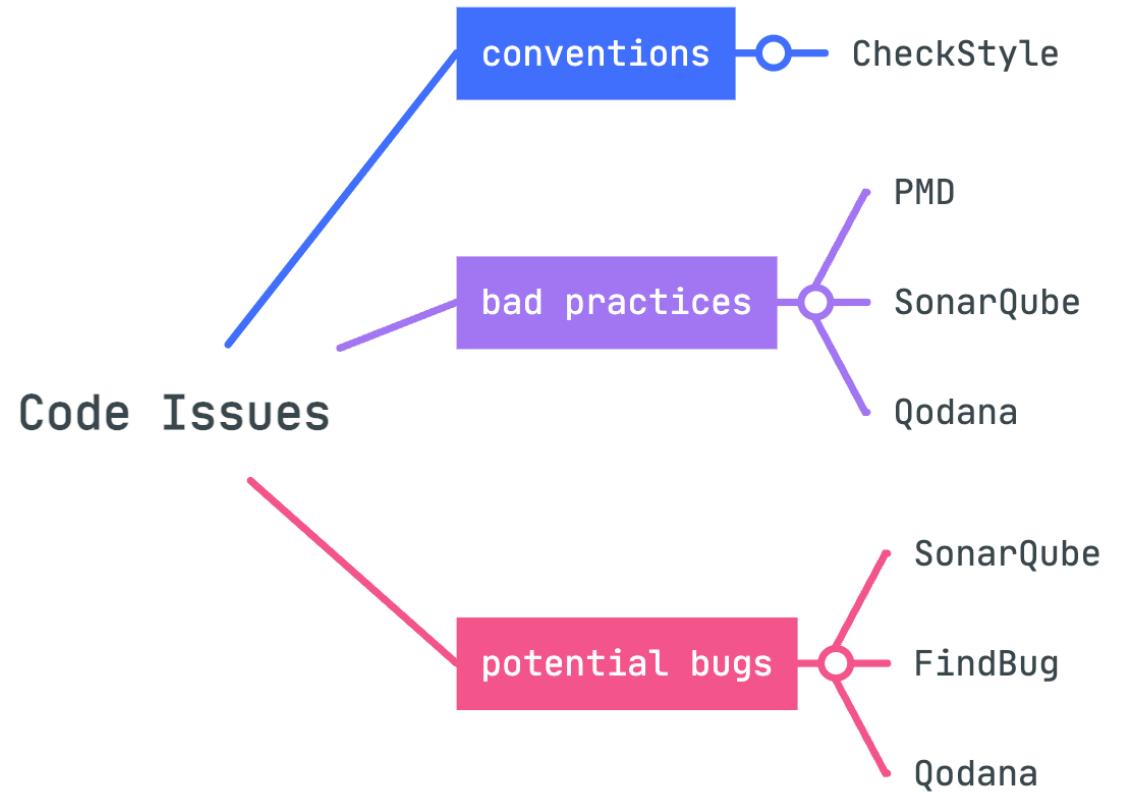
It's as easy as pie to manage your code



when you alone

Context - We have a problem

But it becomes harder when several teams are working on one project



How to solve these issues?

SonarQube



SonarQube

SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs and code smells.

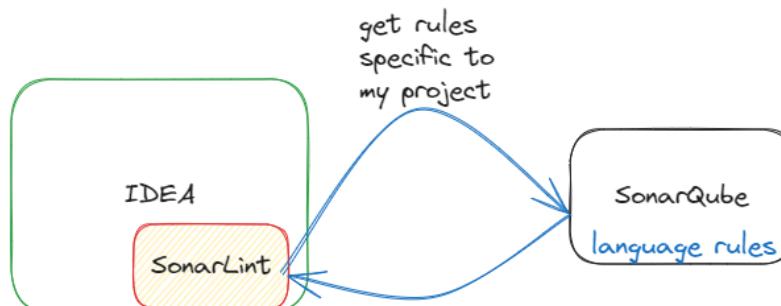
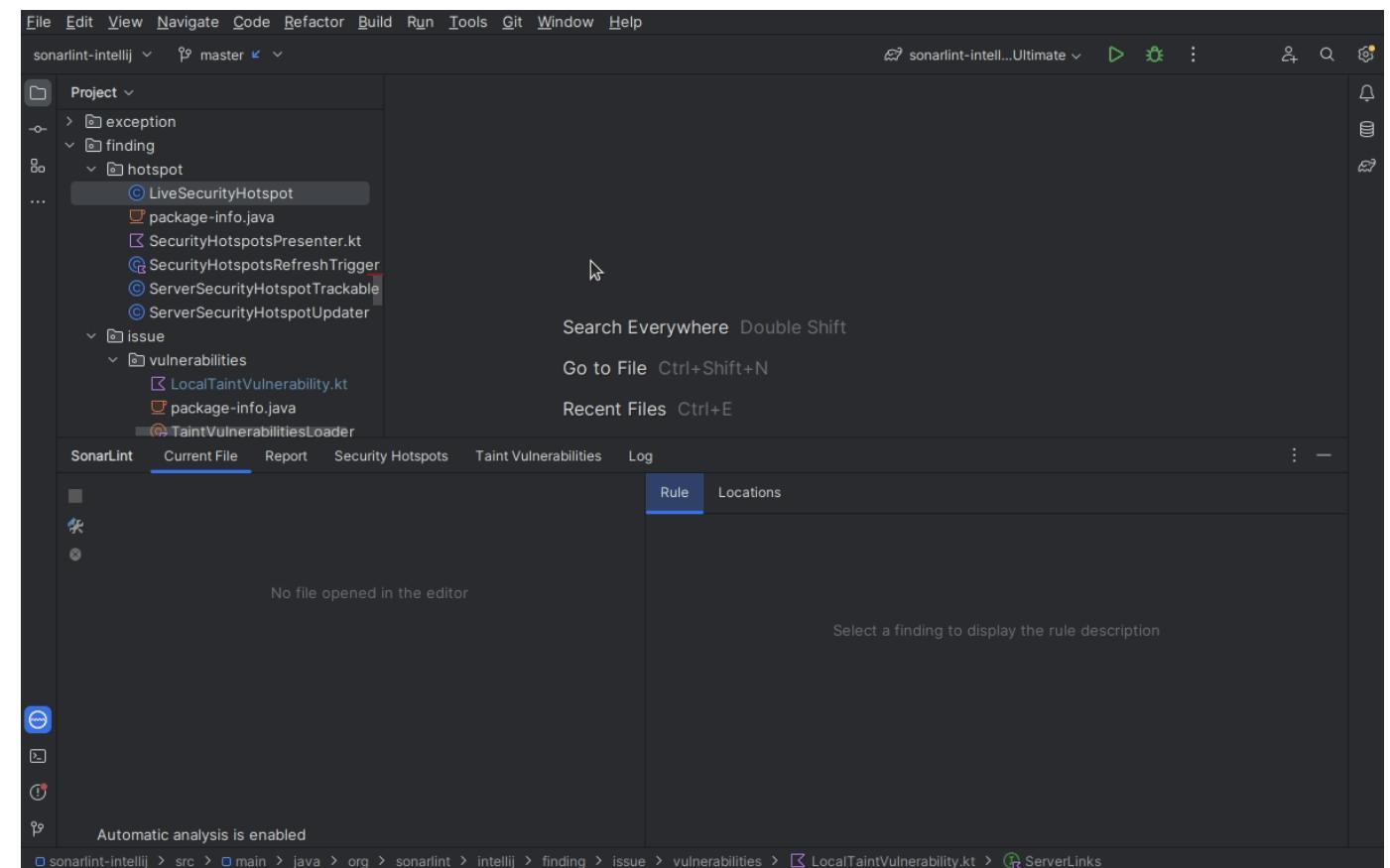


Figure 1: Getting latest language rules from server



SonarQube

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration ? Search for projects, sub-projects and files A

Filters Clear All Filters

Search for rules...

▼ Language Java 320 Search for languages...

▼ Type Bug 100 Vulnerability 26 Code Smell 194

▶ Tag

Bulk Change

↑ ↓ to select rules ← → to navigate 1 / 320 rules

✖ ".equals()" should not be used to test the values of "Atomic" classes	Java ⚡ Bug 🛡 multi-threading	Deactivate
✖ "=+" should not be used instead of "+="	Java ⚡ Bug	Deactivate
❗ "@Controller" classes that use "@SessionAttributes" must call "setComplete" on their "SessionStatus" objects	Java ⚡ Bug 🛡 spring	Deactivate
✖ "@Deprecated" code should not be used	Java ⚡ Code Smell 🛡 cert, cwe, obsolete	Deactivate
✖ "@NotNull" values should not be set to null	Java ⚡ Bug 🛡 cert, cwe	Deactivate
✖ "@Override" should be used on overriding and implementing methods	Java ⚡ Code Smell 🛡 bad-practice	Deactivate
SpringBootApplication" and "@ComponentScan" should not be defined in the default package	Java ⚡ Bug 🛡 spring	Deactivate

- * Code smell (maintainability domain)
- * Bug (reliability domain)
- * Vulnerability (security domain)
- * Security hotspot (security domain)

Sonar Maven Plugin Setup

```
<plugin>
  <groupId>org.sonarsource.scanner.maven</groupId>
  <artifactId>sonar-maven-plugin</artifactId>
  <version>latest</version>
</plugin>
```

1

```
mvn clean verify sonar:sonar -Dsonar.projectKey=PROJECT_KEY
-Dsonar.projectName='PROJECT_NAME'
-Dsonar.host.url=http://localhost:9000
-Dsonar.token=THE_GENERATED_TOKEN
```

2

baeldung.com/sonar-qube

PMD

PMD is a source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It supports Java, JavaScript, Salesforce.com Apex and Visualforce, PLSQL, Apache Velocity, XML, XSL.

baeldung.com/pmd

PMD Setup

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-pmd-plugin</artifactId>
  <version>3.14.0</version>
  <executions>
    <execution>
      <goals>
        <goal>check</goal> <!-- Run in build phase -->
      </goals>
    </execution>
  </executions>
</plugin>
```

1

2

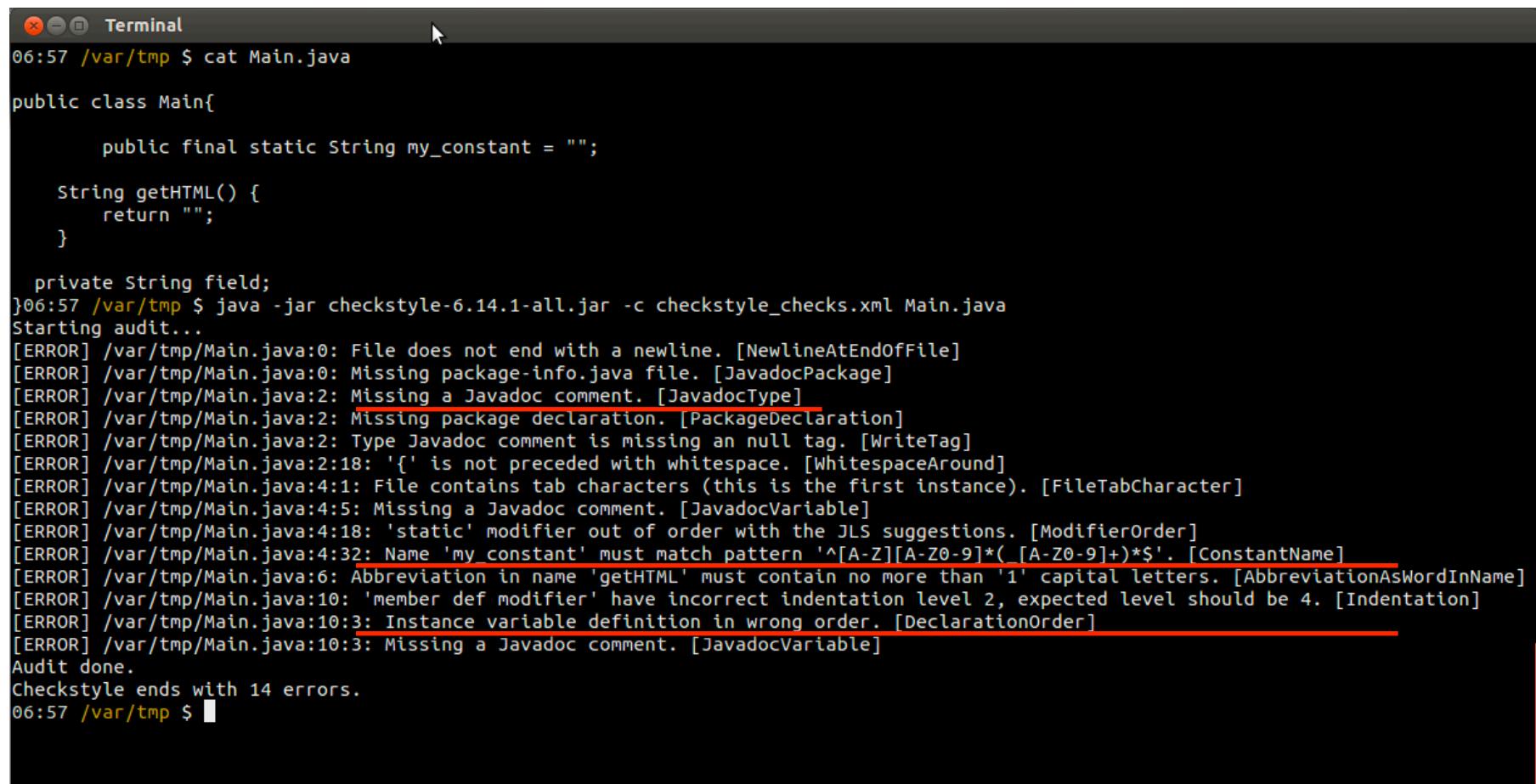
mvn pmd:check

3

```
[INFO] --- pmd:3.19.0:check (default) @ pricing-rewardconversion-service ---
[INFO] PMD version: 6.50.0
[INFO] PMD Failure: com.emirates.ocsl.pricing.*.domain.conversion.MilesDistributionConverter:49
Rule:ExcessiveMethodLength Priority:3 Avoid really long methods..
```

CheckStyle

Checkstyle is a static code analysis tool used in software development for checking if Java source code is compliant with specified coding rules.



A terminal window titled "Terminal" showing the output of a Checkstyle audit on a Java file named Main.java. The terminal shows the Java code, the command to run Checkstyle, and the resulting audit errors. The errors are highlighted with red boxes, indicating specific coding violations.

```
06:57 /var/tmp $ cat Main.java
public class Main{
    public final static String my_constant = "";
    String getHTML() {
        return "";
    }
    private String field;
}06:57 /var/tmp $ java -jar checkstyle-6.14.1-all.jar -c checkstyle_checks.xml Main.java
Starting audit...
[ERROR] /var/tmp/Main.java:0: File does not end with a newline. [NewlineAtEndOfFile]
[ERROR] /var/tmp/Main.java:0: Missing package-info.java file. [JavadocPackage]
[ERROR] /var/tmp/Main.java:2: Missing a Javadoc comment. [JavadocType]
[ERROR] /var/tmp/Main.java:2: Missing package declaration. [PackageDeclaration]
[ERROR] /var/tmp/Main.java:2: Type Javadoc comment is missing an null tag. [WriteTag]
[ERROR] /var/tmp/Main.java:2:18: '{' is not preceded with whitespace. [WhitespaceAround]
[ERROR] /var/tmp/Main.java:4:1: File contains tab characters (this is the first instance). [FileTabCharacter]
[ERROR] /var/tmp/Main.java:4:5: Missing a Javadoc comment. [JavadocVariable]
[ERROR] /var/tmp/Main.java:4:18: 'static' modifier out of order with the JLS suggestions. [ModifierOrder]
[ERROR] /var/tmp/Main.java:4:32: Name 'my_constant' must match pattern '^[A-Z][A-Z0-9]*([A-Z0-9]+)*$'. [ConstantName]
[ERROR] /var/tmp/Main.java:6: Abbreviation in name 'getHTML' must contain no more than '1' capital letters. [AbbreviationAsWordInName]
[ERROR] /var/tmp/Main.java:10: 'member def modifier' have incorrect indentation level 2, expected level should be 4. [Indentation]
[ERROR] /var/tmp/Main.java:10:3: Instance variable definition in wrong order. [DeclarationOrder]
[ERROR] /var/tmp/Main.java:10:3: Missing a Javadoc comment. [JavadocVariable]
Audit done.
Checkstyle ends with 14 errors.
06:57 /var/tmp $
```

CheckStyle Setup

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <version>3.2.0</version>
  <configuration>
    <configLocation>src/main/resources/checkstyle.xml</configLocation>
  </configuration>
  <executions>
    <execution>
      <id>validate</id>
      <phase>validate</phase>
      <goals>
        <goal>check</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

1

2

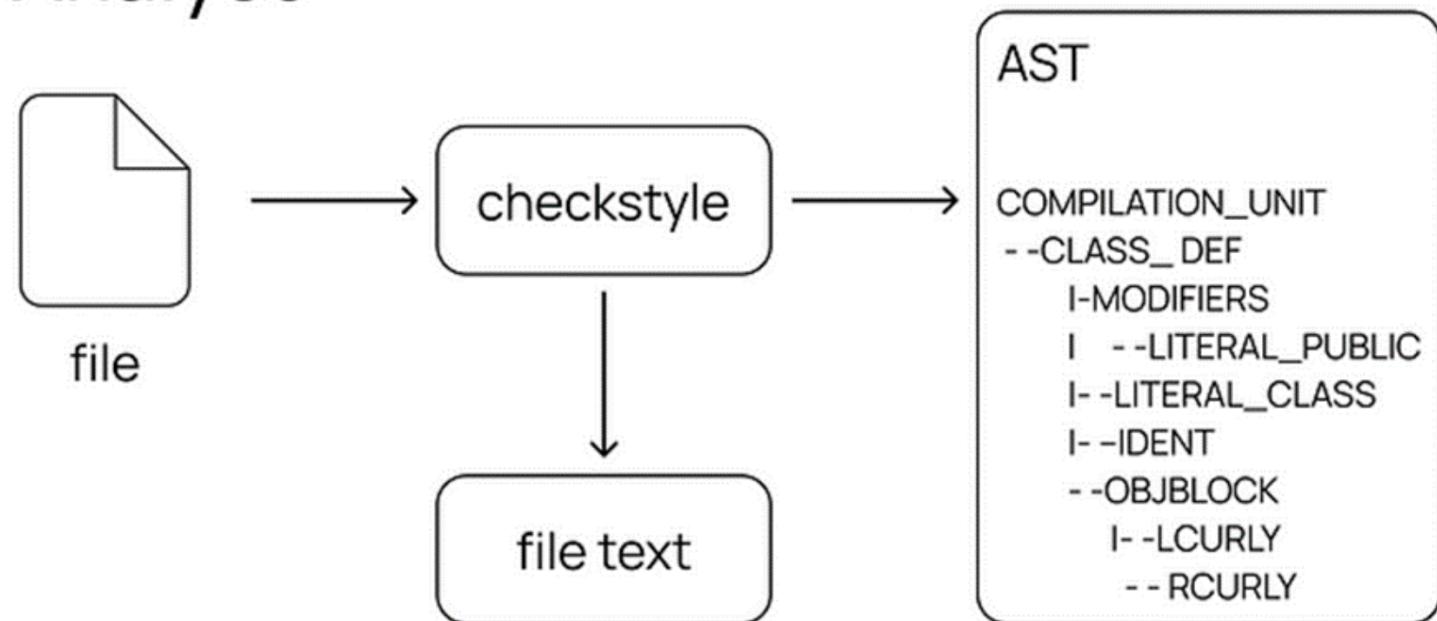
mvn checkstyle:check

3

File.java:10: Missing a Javadoc comment. [JavadocType]
File.java:14: Line is longer than 80 characters (found 85). [LineLength]

How does CheckStyle work?

Analyse



Abstract Syntax Tree (AST)

```
public class Main {
    public static void main(String[] args)
        System.out.println("Hello world!");
}
```



```

CLASS_DEF -> CLASS_DEF [1:0]
|--MODIFIERS -> MODIFIERS [1:0]
|   '--LITERAL_PUBLIC -> public [1:0] 1
|--LITERAL_CLASS -> class [1:7]
|--IDENT -> Main [1:13] 2
`--OBJBLOCK -> OBJBLOCK [1:17] 3
    |--LCURLY -> { [1:17]
    |--METHOD_DEF -> METHOD_DEF [2:4]
        |--MODIFIERS -> MODIFIERS [2:4]
            |--LITERAL_PUBLIC -> public [2:4]
            '--LITERAL_STATIC -> static [2:11]
        |--TYPE -> TYPE [2:18]
            '--LITERAL_VOID -> void [2:18]
        |--IDENT -> main [2:23]
        |--LPAREN -> ( [2:27]
        |--PARAMETERS -> PARAMETERS [2:34]
            '--PARAMETER_DEF -> PARAMETER_DEF [2:34]
                |--MODIFIERS -> MODIFIERS [2:34]
                |--TYPE -> TYPE [2:34]
                    '--ARRAY_DECLARATOR -> [ [2:34]
                        |--IDENT -> String [2:28]
                        '--RBRACK -> ] [2:35]
                    '--IDENT -> args [2:37]
            |--RPAREN -> ) [2:41]
            '--SLIST -> { [2:43]

```

CheckStyle Custom Behavior



RegExp

XPath

Pure Java

RegExp

widely used

>> analyse scripts, configuration files

Examples: check lines, whitespaces and etc

Example (RegExp)

Example #1: You need to check the config files where **each maven command should contain two flags: -e and -no-transfer-progress**

```
<module name="RegexpSingleline">
  <property name="format" value="^#[^#]*mvn (?!( -e -no-transfer-progress))" />
  <property name="fileExtensions" value="sh, yml" />
</module>
```

Example (RegExp)

Example #2: You need to check that **each test class should contain two test methods with @Test**

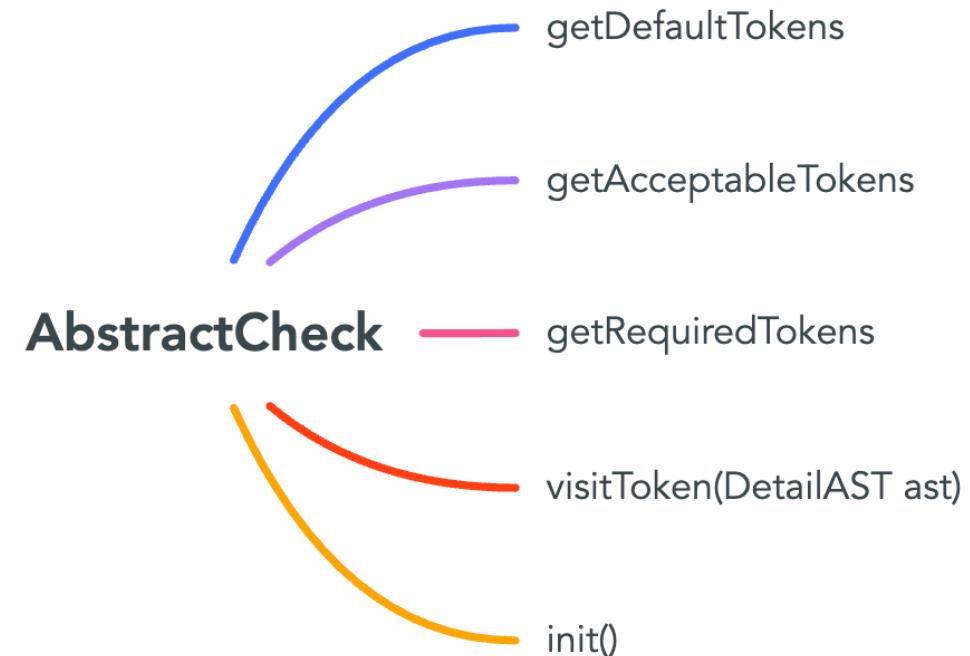
```
<module name="RegexpSingleline">
    <property name="format" value="@Test"/>
    <property name="minimum" value="2"/>
    <property name="fileExtensions" value="java"/>
</module>
```

Example (XPath)

Example #1: You are **not allowed to use @Issue** in your project.

```
<module name="MatchXpath">
  <property name="query" value="//ANNOTATION[ ./IDENT[ @text='Issue' ] ]" />
</module>
```

Pure Java



Example (Pure Java)

```
private DetailAST findLastChildWhichHasType(DetailAST ast, int type) {  
    DetailAST child = ast.getLastChild();  
    while (child != null && child.getType() != type) {  
        child = child.getPreviousSibling();  
    }  
    return child;  
    //empty line  
    //empty line  
}
```



Avoid leaving empty lines at the end of methods

Example (Pure Java)

```
public void visitToken(DetailAST ast) {
    DetailAST openingBrace = ast.findFirstToken(TokenTypes.SLIST);
    DetailAST closingBrace =
        (openingBrace != null) ? findLastChildWhichHasType(openingBrace, TokenTypes.RCURLY)
    : null;

    if (openingBrace != null && closingBrace != null) {
        int open = openingBrace.getLineNo();
        int end = closingBrace.getLineNo();
        checkEmptyLinesAfter(open, end);
        checkEmptyLinesBefore(end, open);
    }
}
```

What should you also know?

Ignore violations:

`<module name="SuppressionFilter">
 <property name="file" value="checkstyle-suppressions.xml"/>
</module>`

1

CheckStyle GUI:

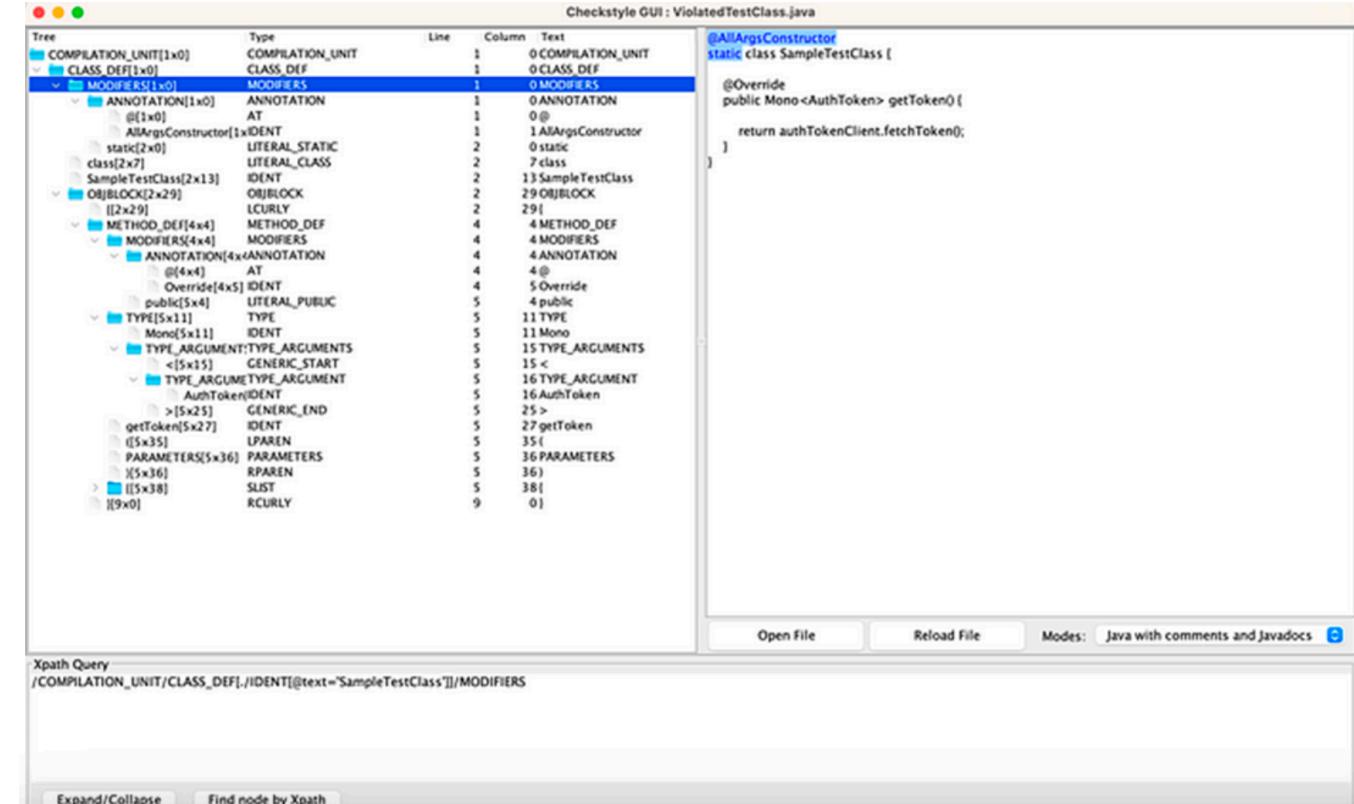
`java -cp checkstyle-10.12.2-all.jar
com.puppycrawl.tools.checkstyle.gui.Main`

2

Additional custom rules:

github.com/sevntu-checkstyle/sevntu.checkstyle
github.com/checkstyle-addons/checkstyle-addons

3



The screenshot shows the Checkstyle GUI interface. On the right, there is a code editor window titled "Checkstyle GUI : ViolatedTestClass.java" containing the following Java code:

```

@AllArgsConstructor
static class SampleTestClass {
    @Override
    public Mono<AuthToken> getToken() {
        return authTokenClient.fetchToken();
    }
}
  
```

On the left, there is a tree view window titled "Tree" showing the parse tree structure of the Java code. The tree structure includes nodes for Compilation Unit, Class Definition, Modifiers, Annotations, Methods, Types, and Parameters.

At the bottom of the interface, there is an "Xpath Query" field with the following expression:

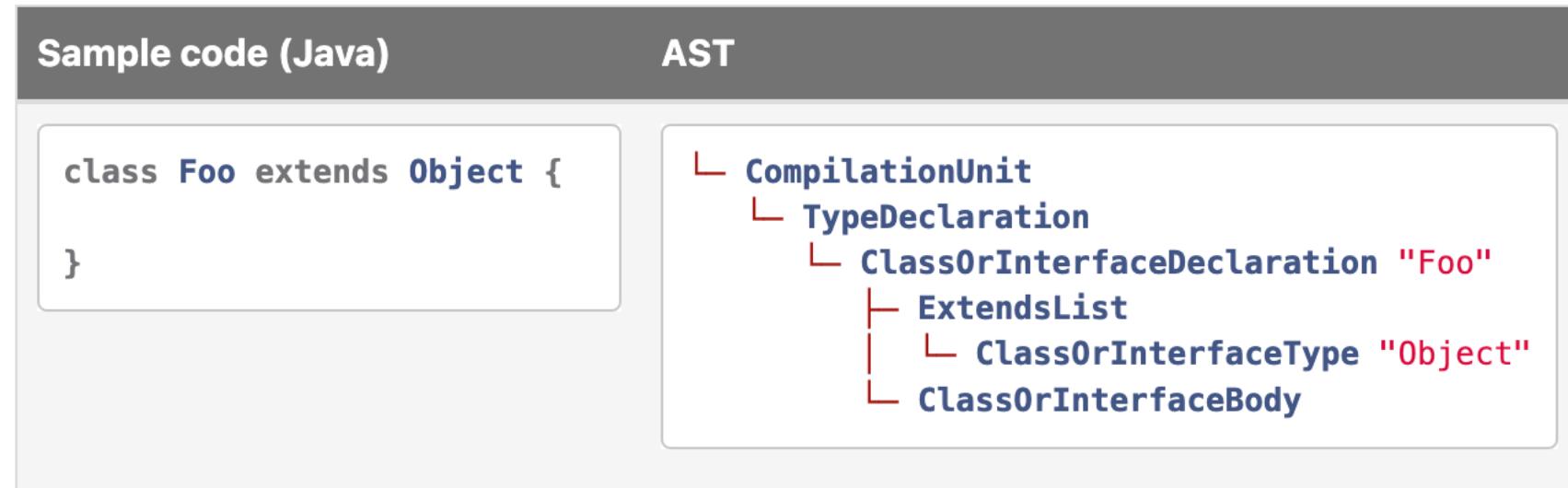
```
/COMPILEATION_UNIT/CLASS_DEF./IDENT[@text='SampleTestClass']/MODIFIERS
```

Below the query field are buttons for "Expand/Collapse" and "Find node by Xpath".

PMD Customization

Before running rules, PMD parses the source file into a data structure called an **abstract syntax tree (AST)**. This tree represents the syntactic structure of the code, and encodes syntactic relations between source code elements.

For instance, in Java, method declarations belong to a class: in the AST, the nodes representing method declarations will be descendants of a node representing the declaration of their enclosing class.



PMD Customization

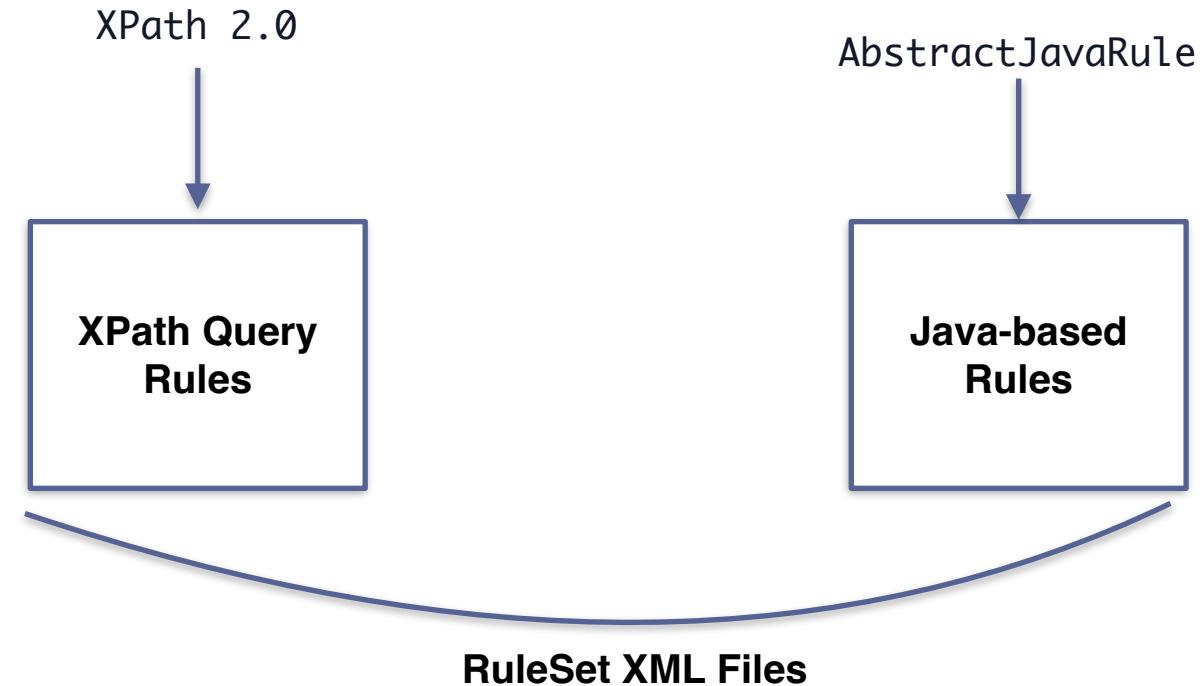
PMD Designer



The screenshot shows the PMD Designer interface with the following components:

- Source Code:** Displays Java code for an enum named Foo. The code includes an annotation @Override at line 4 and a toString() method at line 5.
- Abstract Syntax Tree (AST):** Shows the hierarchical structure of the code. The enum declaration "Foo" is highlighted. The @Override annotation is also visible under the ClassOrInterfaceBodyDeclaration node.
- Navigation:** A bottom navigation bar with tabs: ... (1), TypeDeclaration, EnumDeclaration, EnumBody, ClassOrInterfaceBodyDeclaration, Annotation (highlighted in blue), MarkerAnnotation, and Name.
- Left Sidebar:** Contains sections for Attributes, Metrics (none), and Scopes (none).
- Top Bar:** Includes File, View, About, Language: Java, and XPath Attributes buttons.

PMD Custom Behavior



docs.pmd-code.org/latest/pmd_userdocs_extending_testing.html

Xpath Custom Rules

Mono

1

```
.just("dear")  
  
.switchIfEmpty(Mono.just(getRandomString())); // non-compliant
```

Mono

2

```
.just("dear")  
  
.switchIfEmpty(Mono.error(getException())); //non-compliant
```

Mono

3

```
.just("dear")  
  
.switchIfEmpty(Mono.defer(() -> Mono.just(getRandomString()))); // compliant
```

Mono

4

```
.just("dear")  
  
.switchIfEmpty(Mono.error(() -> getException())); //compliant
```

Xpath Custom Rules

```
<ruleset name="Custom Rules for Emirates Code Base"
  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>
  Rules which enforce generally accepted best practices.
</description>

<rule name="SwitchIfEmptyIncorrectUsage"
  language="java"
  message="Avoid direct call of Mono.switchIfEmpty with non-constant parameter and use Mono.defer instead"
  class="net.sourceforge.pmd.lang.rule.XPathRule">
<description>
  By design Mono.switchIfEmpty is being evaluated eagerly and it's always trigger alternative no matter what during stream
  construction. To address this you can defer evaluation of a second mono by using Mono.defer.
</description>
<priority>1</priority>
<properties>
  <property name="xpath">
    <value>
      <![CDATA[
//PrimarySuffix[@Image = "switchIfEmpty"]
[
(following-sibling::PrimarySuffix/
 Arguments/
 ArgumentList/
 Expression/
 PrimaryExpression/
 PrimaryPrefix/
 Name/@Image != "Mono.defer"
and
following-sibling::PrimarySuffix/
 Arguments/
 ArgumentList/
 Expression/
 PrimaryExpression/
 PrimaryPrefix/
 Name/@Image != "Mono.defer"
      ]]</value>
    </property>
  </properties>
</rule>
```

Pay attention



Pay attention



Pure Java Custom Rules

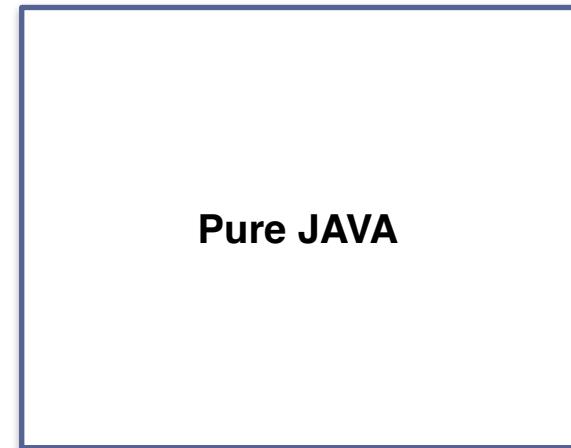
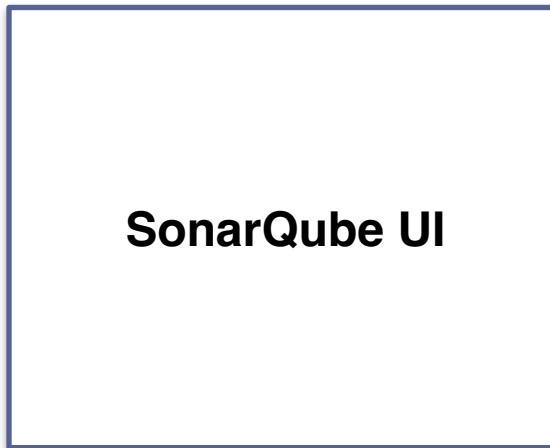
Test must have TestId annotation

```
@ParameterizedTest  
@ValueSource(strings = {"apple", "banana"})  
@TestCaseId(2)  
void testValueSource(String fruit) {  
    assertNotEquals("pineapple", fruit);  
}
```

Solution

```
public class TestMustHaveTestIdAnnotationRule extends AbstractJavaRule {  
    @Override 2  
    public Object visit(final ASTMethodDeclaration node, final Object data) {  
        ASTAnnotation testAnnotation = node.getAnnotation(Test.class.getCanonicalName());  
        ASTAnnotation parameterizedTestAnnotation =  
node.getAnnotation(ParameterizedTest.class.getCanonicalName());  
  
        if (testAnnotation != null || parameterizedTestAnnotation != null) {  
            ASTAnnotation testCaseIdAnnotation =  
node.getAnnotation(TestCaseId.class.getCanonicalName()); 3  
            if (testCaseIdAnnotation == null) {  
                addViolation(data, node);  
            } 4  
        }  
        return super.visit(node, data);  
    }  
}
```

How to create a Custom SonarQube Rule?



<https://dilumpathi.medium.com/adding-sonarqube-coding-rules-using-java-for-the-test-automation-suite-c4083405278a>

https://github.com/SonarSource/sonar-javascript/blob/master/docs/CUSTOM_RULES_101.md

Pure Java

1

```
<dependency>
  <groupId>org.sonarsource.sonarqube</groupId>
  <artifactId>sonar-plugin-api</artifactId>
  <version>${sonarqube.version}</version>
  <scope>provided</scope>
</dependency>
```

MyJavaRulesPlugin implements Plugin	<i>Entry point of your plugin containing your custom rules</i>
MyJavaRulesDefinition implements RulesDefinition	<i>Declare rule metadata in server repository of rules. That allows to list the rules in the page "Rules".</i>
MyJavaFileCheckRegistrar implements CheckRegistrar	<i>Provide the checks (implementations of rules) classes that are going be executed during source code analysis.</i>
@Rule(key = "MyFirstCustomRule") public class MyFirstCustomCheck extends IssuableSubscriptionVisitor	<i>Override nodesToVisit and visitNode</i>

2

3

Put your jar to Sonar instance extensions/plugins

EK Payments

- 1 We have a bom file
- 2 We have separate dependency specific to PMD, CheckStyle rules
- 3 We use bom in each pom services
If needed we override parameters for each plugin
- 4 We wouldn't allow to build PR with violations

EK Payments Customization

PMD	CheckStyle	SonarQube
Basic Rules Set	Basic Rules Set	Basic Rules for each language
+ Custom Rules	+ Custom Rules (JavaDoc for new code in PRs)	

Conclusion

1. It was a brief overview of PMD, CheckStyle, SonarQube
2. Customization PMD (Xpath, PureJava - AbstractJavaRule)
3. We have looked at how we could customize CheckStyle (Xpath, RegExp, AbstractCheck)
4. We have looked at how we could customize SonarQube (UI, plugin system)
5. We have looked at how we adopted these tools to URP projects in EK
- 6*. Live Demonstration (time permitting)

QA

1. <https://www.sonarsource.com/blog/what-makes-checkstyle-pmd-findbugs-and-macker-complementary/>
2. <https://www.linkedin.com/pulse/customize-checkstyle-rules-your-development-process-artem-sokovets-kxuyf/>
3. <https://medium.com/@rahul.gite11/checkstyle-all-you-need-to-know-5b6e7a37072a>
4. <https://medium.com/@norsag/custom-pmd-rules-488cbd332ad9>
5. <https://github.com/NorsaG/evla-custom-pmd/blob/master/src/main/resources/rules.xml>
6. <https://datamify.medium.com/java-pmd-how-to-check-java-source-code-with-pmd-2df2783e9f30>
7. <https://github.com/pmd/pmd-examples/tree/main/maven/simple-project>
8. https://docs.pmd-code.org/pmd-doc-6.55.0/pmd_userdocs_extending_designer_reference.html
9. <https://dilumpathi.medium.com/adding-sonarqube-coding-rules-using-java-for-the-test-automation-suite-c4083405278a>
10. https://github.com/SonarSource/sonar-javascript/blob/master/docs/CUSTOM_RULES_101.md
11. <https://medium.com/@artsokovets/customize-checkstyle-rules-for-your-development-process-bbf7b60c05a5>

Live Demo*

github.com/NorsaG/evla-custom-pmd

github.com/hadiinz/sonar-custom-rule-training

github.com/artsok/custom-checkstyle-rules

- 
- [MissingJavaDocMethodUrpCheck](#)
 - [IntermediateEmptyLinesCheck](#)
 - [EmptyLineAtTheEndChecker](#)
 - [AnnotationsWithoutEmptyLinesChecker](#)