ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTY OF ELECTRICAL AND COMPUTER ENGINEERING
SOFTWARE ENGINEERING II
CODING STANDARDS - I TERM 2021

Objectives:

- Use Checkstyle as a code quality metric tool to detect and enforce Java coding standards.
- Obtain a html report with the different detected issues in the code using Maven.

Requirements

- Eclipse IDE for Java Developers >= 2021-03
- Git

Introduction

Checkstyle is an open-source tool that enforces coding conventions and best practice rules for Java code. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard. [1]

Checkstyle can check many aspects of your source code. It can find class design problems, method design problems. It also can check code layout, formatting issues and generate project-wide reports that summarize the breaches found. Checkstyle provides many checks that you can apply to your source code [1]:

- Annotations
- Block Checks
- Class Design
- Coding
- Headers
- Imports
- Javadoc Comments
- Metrics
- Miscellaneous
- Modifiers
- Naming Conventions
- Regexp
- Size Violations
- Whitespace

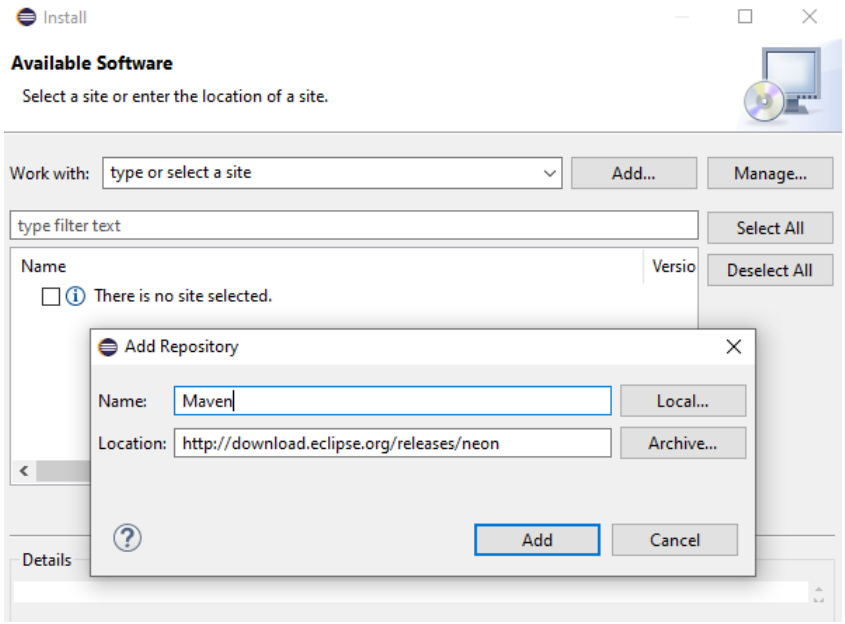
There are several ways to use this tool, including the command line and build automation tools. In this lab, Eclipse will be used in conjunction with Maven to facilitate the use of Checkstyle.

Activity

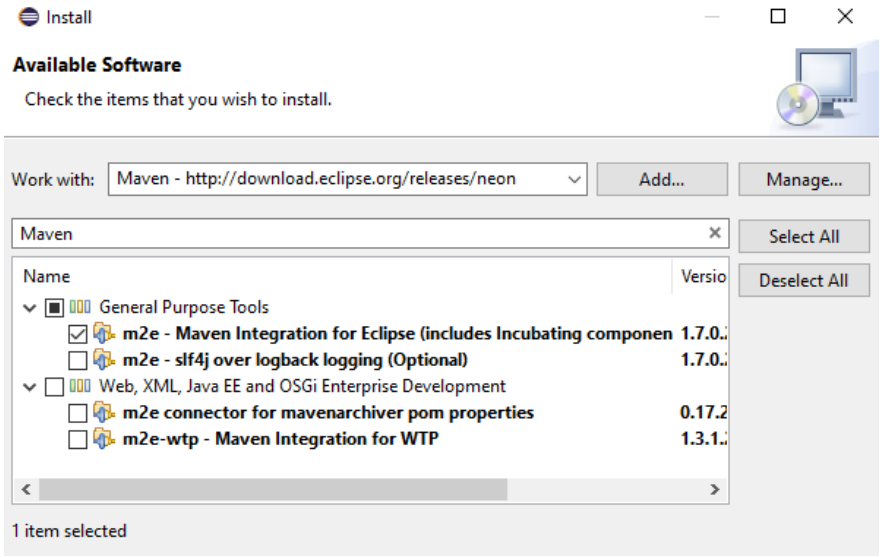
Part 1: Install Maven for Eclipse.

Most Eclipse download include the maven tooling already. If it is missing in your installation, follow these steps, otherwise jump to part two:

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



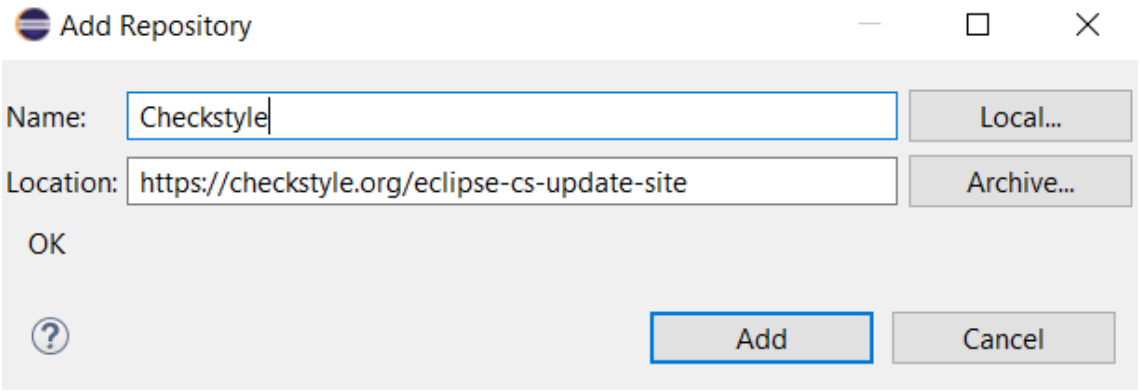
3. Click "Add" again and wait until the process finishes.
4. Search Maven and check "Maven Integration for Eclipse" under "General Purpose Tools"



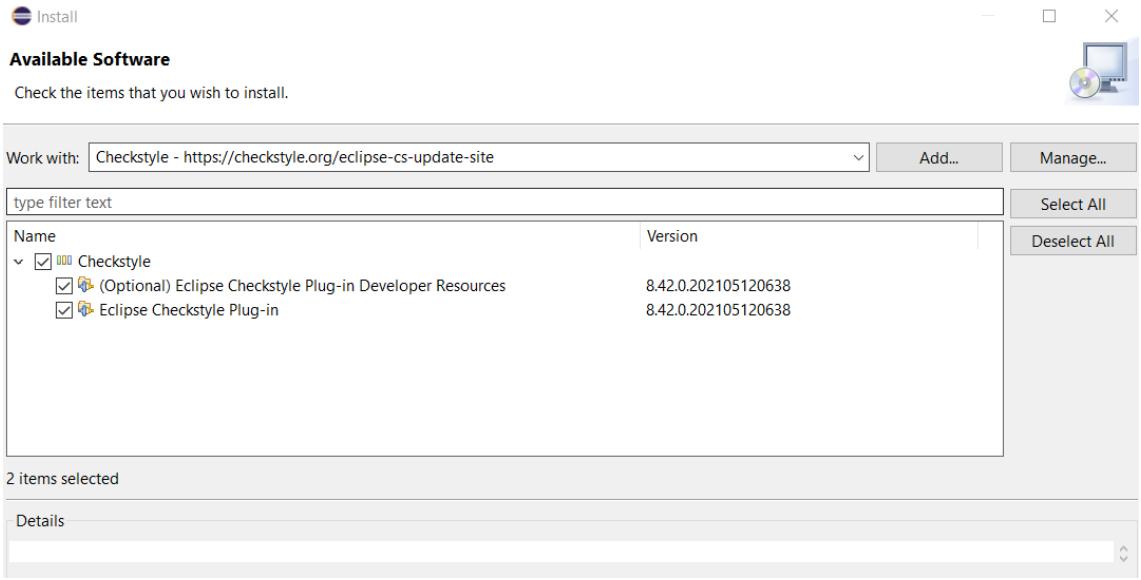
5. Click "Next" and step through the following installation screens.

Part 2: Install the Eclipse Checkstyle plug-in

1. Open the plugin installation window by selecting the "help → Install new software"
2. Click on aAdd and type Checkstyle for the name and https://checkstyle.org/eclipse-cs-update-site for the location.



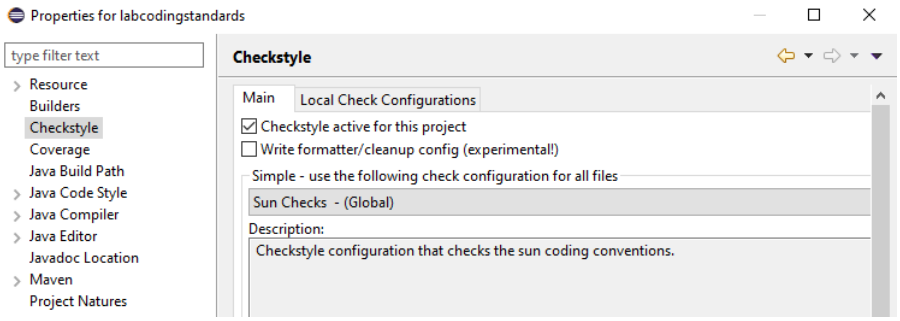
3. Click "Add" again and wait until the process finishes.
4. Check "Checkstyle" and "Extension for eclipse-cs plugin with additional Checks"



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

Part 3: Download and configure the project

1. Clone the following repository: https://github.com/leortyz/CodingStandards and open the project in eclipse.
2. By default, Checkstyle will not be activated for the project. Open the project properties window by clicking in "Project → Properties".
3. Select "Checkstyle" on the side bar.
4. Check "Checkstyle activate for this project "and select "Sun checks – (Global)" on Simple.



Sun Checks [2] and Google Checks [3] are styles configurations for Checkstyle. For more information about the conventions and style, check the corresponding reference.

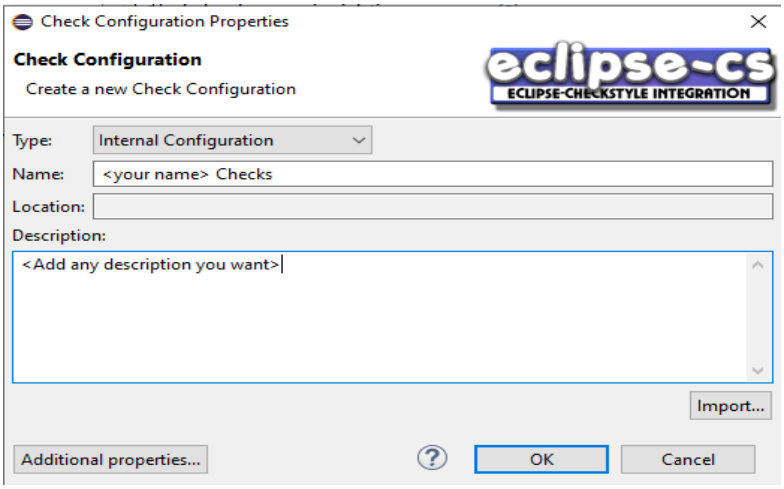
5. Look at the options to exclude from checking. For more information, check chapter 21 section 2 of the book. [1]
6. Click “Apply and Close” and “Yes”.
7. Checkstyle runs as a background task and audits the source code in the project. This may take some time, depending on the size of the project.

Part 4: Configure Checkstyle

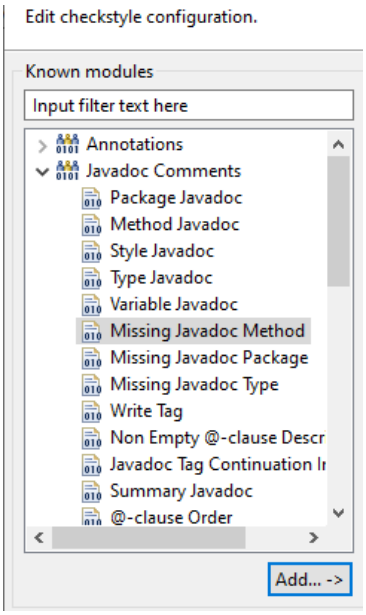
The first thing to note is that the files in the project will be marked with errors. Those errors are the result of Checkstyle doing its work by using the Sun Coding Standards. Take some time to look at all the warnings.

Checkstyle gives the option to create your own custom set of coding standards that is specifically design for your project. To do so, follow these steps:

1. Go to "Window → Preferences "and click "Checkstyle" on the side menu.
2. Check "Include rule names in violation messages" in General Settings
3. Click "New" to start creating a configuration file.
4. Select "Internal Configuration" under Type. For more information about the different types, refer to Chapter 21 Section 3 of the book [1].
5. For the Name, type "<your name> Checks" and add a Description.



6. Click on "Ok" to create the file
7. The configuration file will be ready to add new rules. Double click on it or click "Configure".
8. All available Checkstyle modules are displayed in groups and are ready to be added to the project. For more information on groups and modules, refer to Checkstyle [4]
9. Let's add some rules. Under "Javadoc Comments", click "Missing Javadoc Method", read the description, then click "Add… ->"



10. Look at all the configuration options the module has. Without modifying, click “Ok”.
11. Now add a new rule for indentations. Under “Indentation”, click “Indentation” then click “Add… ->”
12. Leave the default options, click “Ok”.

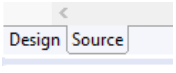
In case you add a wrong rule or misconfigure it, you can do the following

- To delete a rule, select an added rule, and then click "<- Delete".
- To enable/disable a rule, select an added rule and then check/uncheck Enabled. Note that the rule will not be deleted, it will only be ignored in the checking process.

13. If you are happy with the results, click “Ok”
14. To export the file, select your check configuration in the Checkstyle configuration screen and click “Export” button and save it somewhere on your hard disk.
15. The configuration file is an XML file. Now you can publish it to your repository.
16. Check again the code and see the new warnings produced by Checkstyle.

Part 5: Configure Checkstyle using the XML

1. Right click on the project, then “New → Other → XML → XML File”.
2. Select the project, enter a file name as “<your name>_ checks “
3. Click “Source” in the bottom tab to change the view and edit the file directly



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

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC
  "-//Checkstyle//DTD Checkstyle Configuration 1.3//EN"
  "https://checkstyle.org/dtds/configuration_1_3.dtd">
<module name="Checker">
            <module name="TreeWalker">
            </module>
</module>
```

Please refer to chapter 21 section 4 of the book [1] and the Checkstyle website [4], for more information about the configuration structure file.

5. Let's add a rule to check the patterns in **local variables** declaration. Write this under module "TreeWalker":

```
<module name="LocalVariableName">
        <property name="format" value="^[a-z](_?[a-zA-Z0-9]+)*$" />
</module>
```

This rule checks for names that begin with a lower-case letter, followed by letters, digits, and underscores.

6. Now, add a rule to check for missing methods javadoc. Write the following under "TreeWalker":

```
<module name="MissingJavadocMethod ">
        <property name="scope" value="private"/>
        <property name="allowMissingPropertyJavadoc" value="true" />
</module>
```

This rule checks for method javadoc ignoring getters and setters and including private methods.

7. Finally, add a rule to check for an @author tag:

```
<module name="JavadocType">
        <property name="authorFormat" value="\S" />
</module>
```

This rule might be unnecessary, but sometimes it is important to know the author. It uses the "\S" regexp notation to indicate that a nonempty string is required

Note that it is possible to configure the properties for each module we add. There are several properties, each with its own purpose. To learn more about modules and their properties, refer to the Checkstyle website [4]

For a guideline about which rules to keep and which ones to discard, refer to section 5 chapter 21 of the book [1]

Part 6: Check for headers.

Many companies and projects use a standard file header convention. There are 2 ways to check for headers using Checkstyle: fixed headers or modifiable headers.

Let's add a rule to check for a fixed header. Copy and paste the following under "Checker" and above "TreeWalker":

```
<module name="Header">
    <property name="header" value="// Copyright (C) 2020\n// All rights reserved"/>
</module>
```

If the header needs to be changed in certain cases or a more sophisticated header is needed, please check Chapter 21 Section 5 of the book [1] for more details.

The final file should look like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC
  "-//Checkstyle//DTD Checkstyle Configuration 1.3//EN"
  "https://checkstyle.org/dtds/configuration_1_3.dtd">

<module name="Checker">
    <module name="Header">
        <property name="header"
            value="// Copyright (C) 2020\n// All rights reserved" />
    </module>
    <module name="TreeWalker">
        <module name="LocalVariableName">
            <property name="format" value="^[a-z](_?[a-zA-Z0-9]+)*$" />
        </module>
        <module name="MissingJavadocMethod">
            <property name="allowMissingPropertyJavadoc" value="true" />
        </module>
        <module name="JavadocType">
            <property name="authorFormat" value="\S" />
        </module>
    </module>
</module>
```

Part 7: Suppressing Checkstyle Tests

There will be times when you come across a genuine reason for violating a coding standard for a section of code. The easiest way to deal with cases like this is to use the "SuppressionCommentFilter" module.

1. Go to "Window → Preferences" and double-click your configuration file.
2. Search for "Suppression Comment Filter", select it and click "Add… ->"
3. Leave the default options, add the module and apply your changes
4. Go to the Calculator file and write "CHECKSTYLE: OFF" and "CHECKSTYLE: ON" around main method.

```
//CHECKSTYLE:OFF
    public static void main(String[] args) {
//CHECKSTYLE:ON
```

If you remember, in Part 4, Step 11, we added a rule to verify indentation in all the files. The rule in that part of the code will be ignored in the report and no warning is shown. The other way to suppress rules is by using an XML file, but it is out of the scope of this lab. For detailed information and examples, refer to Section 6, Chapter 21 of the book [1]

Part 8: Checkstyle with Maven

Checkstyle integrates well with Maven, but first we need to add the necessary plugins.

1. Go to the project tree and open "pom.xml"

```xml
<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>labcodingstandards</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>LabCodingStandards</name>
    <description>Lab Coding Standards, Sofware Engineering II</description>

</project>
```

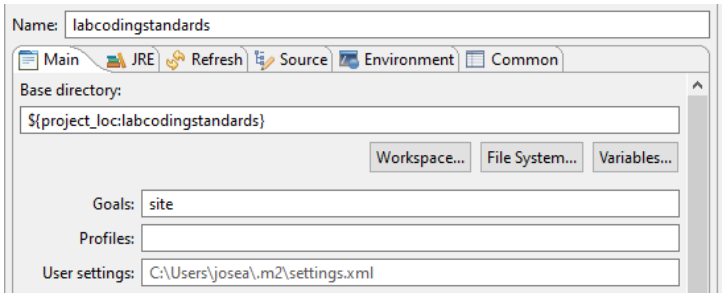
2. Set up the basic configuration of your pom.xml. Add the following lines under project tag:

```xml
<build>
  <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>
    </plugins>
  </build>
  <reporting>
    <plugins>
      <plugin>
                <artifactId>maven-checkstyle-plugin</artifactId>
                      <configuration>
                        <configLocation><yourname>_checks.xml</configLocation>
                      </configuration>
              </plugin>
    </plugins>
  </reporting>
```

The configLocation tag is used to specify what file to use in the checking process. If omitted, the selected file in the Checkstyle configuration window will be used.

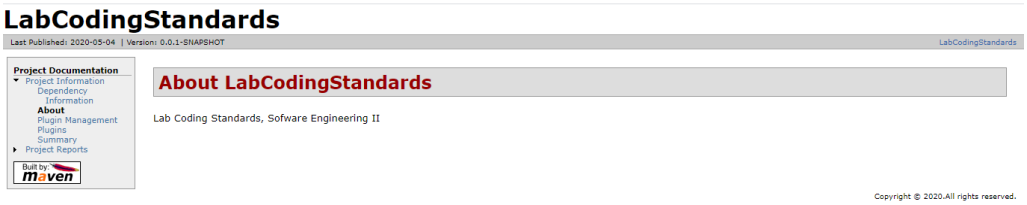
3. Save the file, right Click on it, then "Run as → Maven build…"
4. Type "site" for the Goals and Click Run.



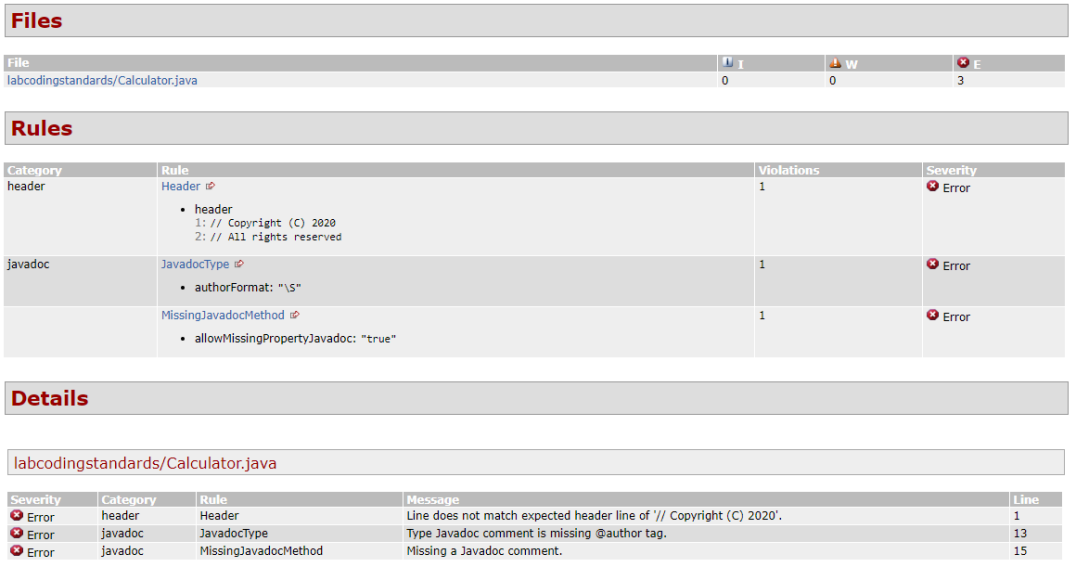
5. Wait for the process to finish.

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  36.259 s
[INFO] Finished at: 2020-05-04T12:15:43-05:00
[INFO] ------------------------------------------------------------------------
```

6. Go to your project directory, open "target→ site"
7. Several html files are show. Open "index.html".



8. Click on "Project Reports → Checkstyle" to see a detailed Checkstyle report of your project.



Files

File	I	W	E
labcodingstandards/Calculator.java	0	0	3

Rules

Category	Rule	Violations	Severity
header	Header • header 1: // Copyright (C) 2020 2: // All rights reserved	1	Error
javadoc	JavadocType • authorFormat: "\S"	1	Error
	MissingJavadocMethod • allowMissingPropertyJavadoc: "true"	1	Error

Details

labcodingstandards/Calculator.java

Severity	Category	Rule	Message	Line
Error	header	Header	Line does not match expected header line of '// Copyright (C) 2020'.	1
Error	javadoc	JavadocType	Type Javadoc comment is missing @author tag.	13
Error	javadoc	MissingJavadocMethod	Missing a Javadoc comment.	15

What you must do for this workshop:

1. Add the following modules and properties to your XML configuration file:
   - Javadoc Method.
   - Method Name, configure a property to match names that begin with a lower-case letter, followed by letters, digits, and underscores.
   - Whitespace After, configure a property to check for whitespace only after COMMA and SEMI tokens.

   TIP: Refer to the Checkstyle website [4]

2. Generate a report, make a copy and save it somewhere on your disk.
3. Correct any errors/warnings in the code that have been generated in the report.
4. Generate a new report. It should not have any errors/warnings.

Deliverables

1. Lab report with screenshots of the process.
2. Two Checkstyle reports.
3. Include in the report the **url** of the repository where you performed the lab.

Rubric

Description	**Value**
Project code (in the repository)	50
Lab report	50
Penalty per hour or fraction of delay.	-30
Penalty for not uploading required deliverables as specified	-30

References

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

[2] Oracle, "Java Code Conventions," 12 September 1997. [Online]. Available: https://www.oracle.com/technetwork/java/codeconventions-150003.pdf.

[3] Google, "Google Java Style Guide," 23 May 2018. [Online]. Available: https://google.github.io/styleguide/javaguide.html.

[4] Checkstyle, "Standard Checks," [Online]. Available: https://checkstyle.sourceforge.io/checks.html.