POLYTECHNIC SCHOOL OF THE LITTORAL

FACULTY OF ELECTRICITY AND COMPUTING

Subject: Software Engineering 2 **Parallel:** 2

Teacher: Eng. Mónica Katiuska Villavicencio Cabezas

Group: 8

Members:

- Joselyne Dayse Maroto Motto
- Daniel Roberto Sanchez Jarrin
- Alexander Alzate Quintero

Start Date: June 3, 2021

End Date: June 4, 2021

Coding Standars

Enlace de Github:

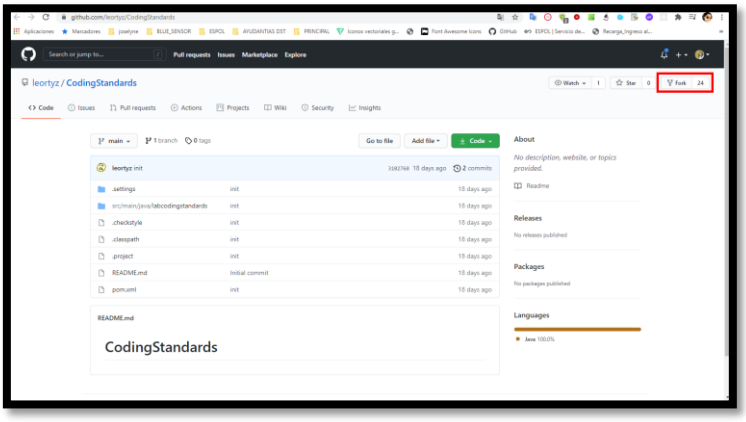
https://github.com/dmaroto98/CodingStandards.git

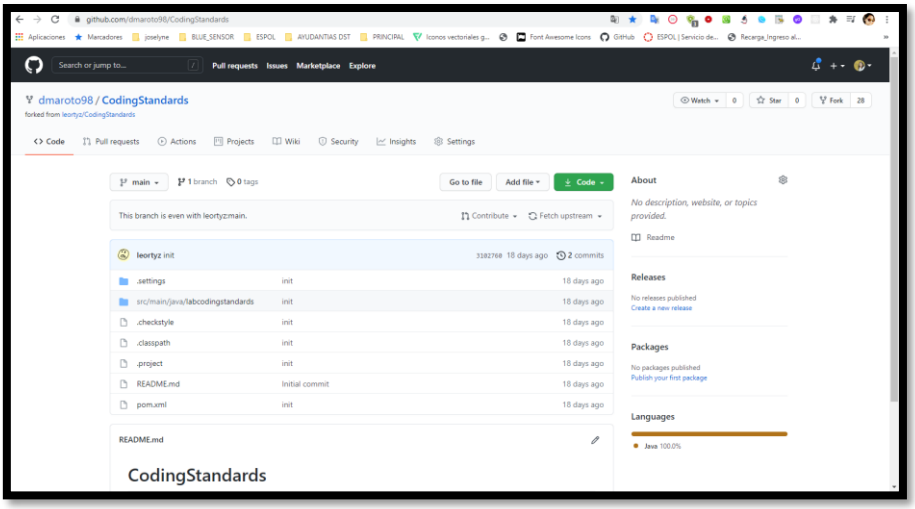
Para este taller se debe tener instalado Maven y el plug-in de checkstyle en eclipse, puede ayudarse con este tutorial para su instalación, parte 1 y parte 2: https://github.com/leortyz/softwareEngineeringResources/wiki/Coding-Standars

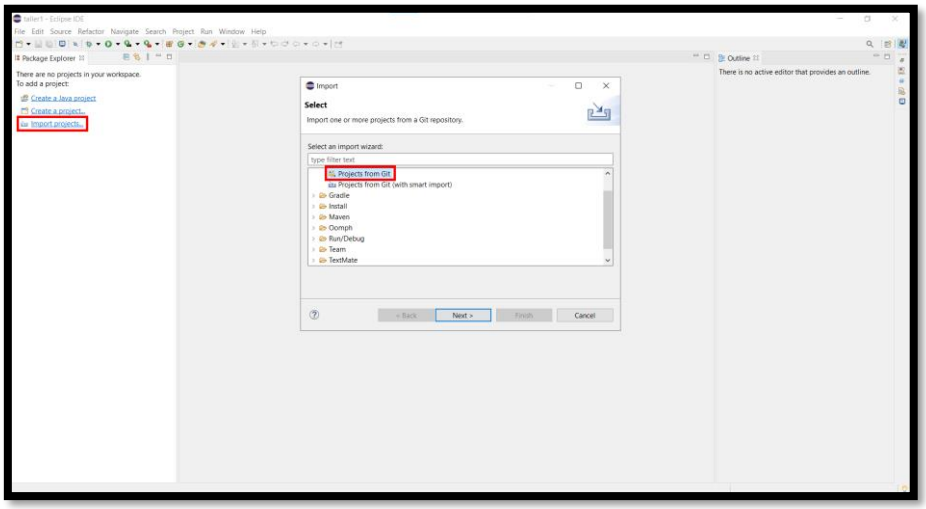
1) Results of activities

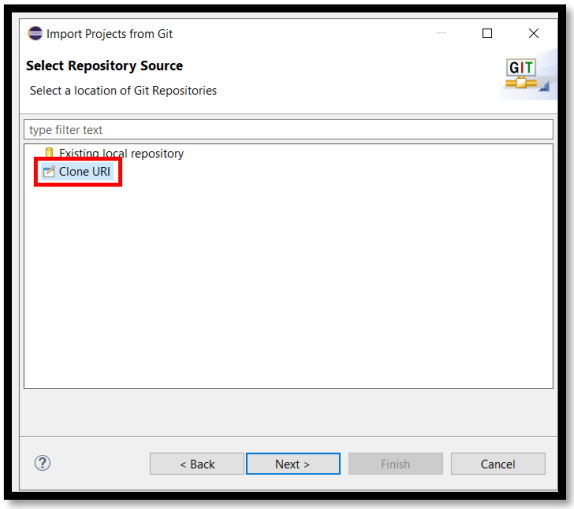
Part 3: Download and configure the project

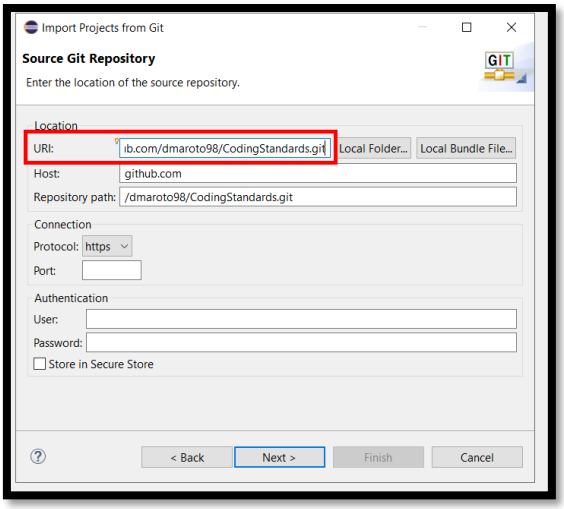
1. Fork the repository CodingStandards and open the project in eclipse.

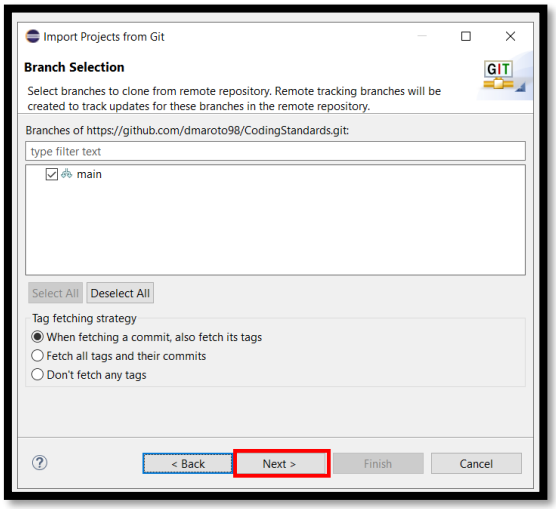


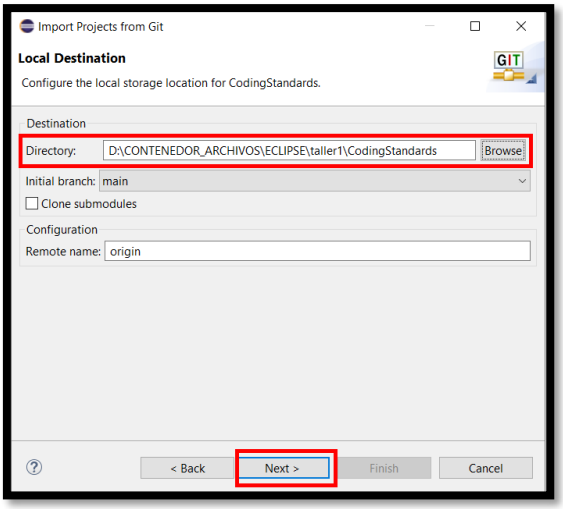

dmaroto98 / CodingStandards
CodingStandards

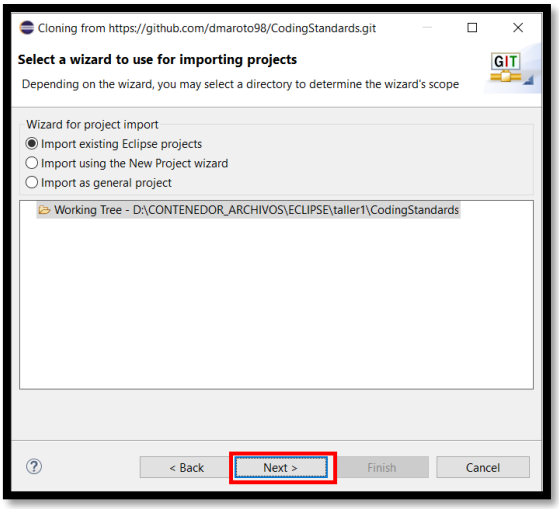

Import
Select
Import one or more projects from a Git repository.
Projects from Git
Projects from Git (with smart import)
Next >
Cancel

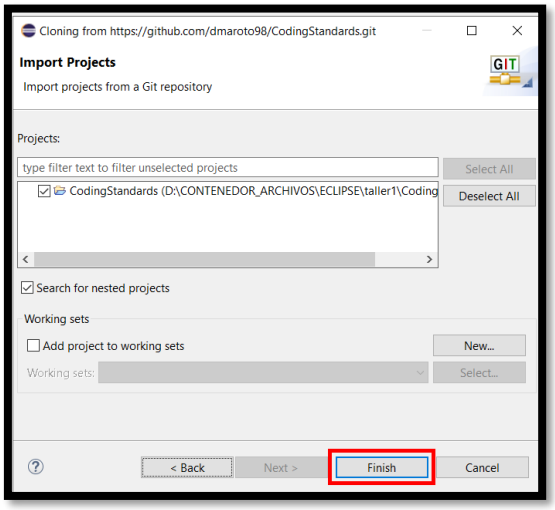

Import Projects from Git
Select Repository Source
Select a location of Git Repositories
type filter text
Existing local repository
Clone URI
< Back
Next >
Finish
Cancel


Import Projects from Git
Source Git Repository
Enter the location of the source repository.
Location
URI:
ib.com/dmaroto98/CodingStandards.git
Local Folder...
Local Bundle File...
Host:
github.com
Repository path:
/dmaroto98/CodingStandards.git
Connection
Protocol:
https
Port:
Authentication
User:
Password:
Store in Secure Store
< Back
Next >
Finish
Cancel

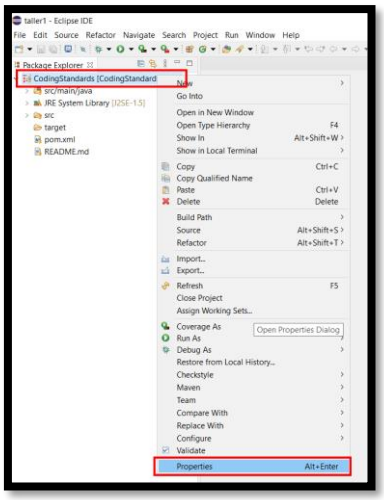

Import Projects from Git
Branch Selection
Select branches to clone from remote repository. Remote tracking branches will be created to track updates for these branches in the remote repository.
Branches of https://github.com/dmaroto98/CodingStandards.git:
type filter text
main
Select All
Deselect All
Tag fetching strategy
When fetching a commit, also fetch its tags
Fetch all tags and their commits
Don't fetch any tags
< Back
Next >
Finish
Cancel


Import Projects from Git
Local Destination
Configure the local storage location for CodingStandards.
Destination
Directory:
D:\CONTENEDOR_ARCHIVOS\ECLIPSE\taller1\CodingStandards
Browse
Initial branch:
main
Clone submodules
Configuration
Remote name:
origin
< Back
Next >
Finish
Cancel

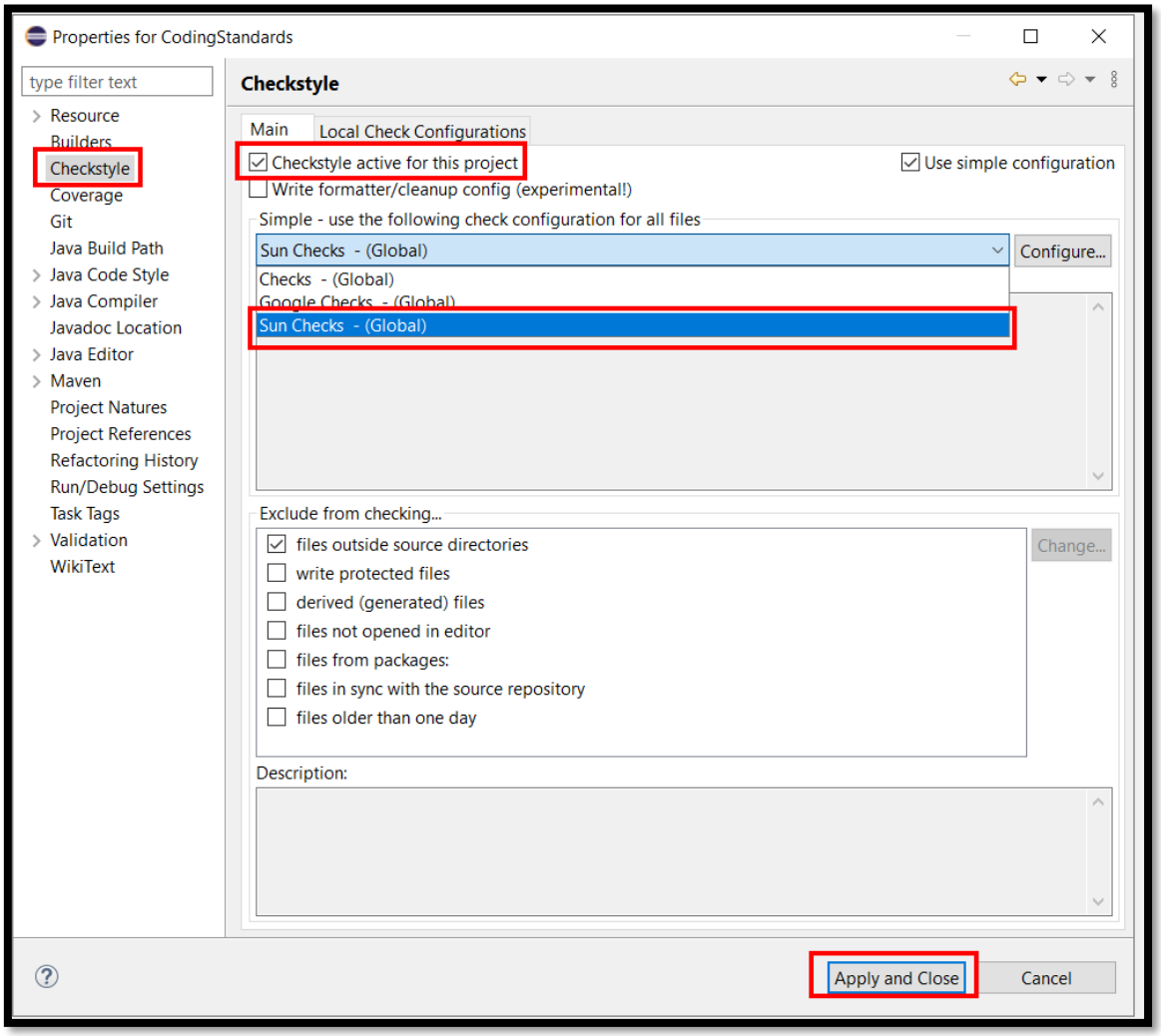




2. By default, Checkstyle will not be activated for the project. Open the project properties window by clicking in “Project > Properties”.



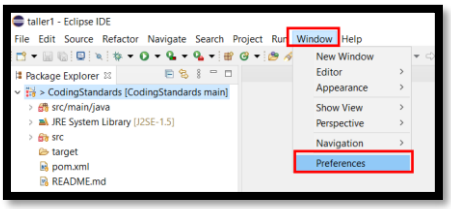
3. Select “Checkstyle” on the side bar.
4. Check “Checkstyle activate for this project “and select “Sun checks – (Global)” on Simple.
   Sun Checks [2] and Google Checks [3] are styles configurations for Checkstyle. For more information about the conventions and style, check the corresponding reference.
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.



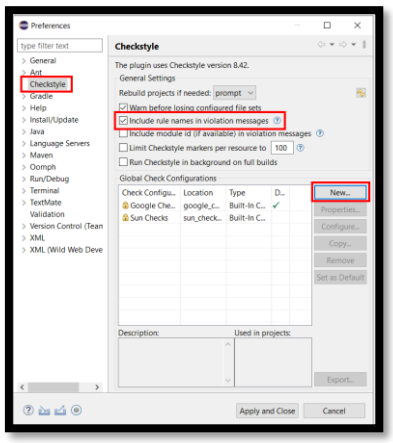
Part 4: Configure Checkstyle

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:

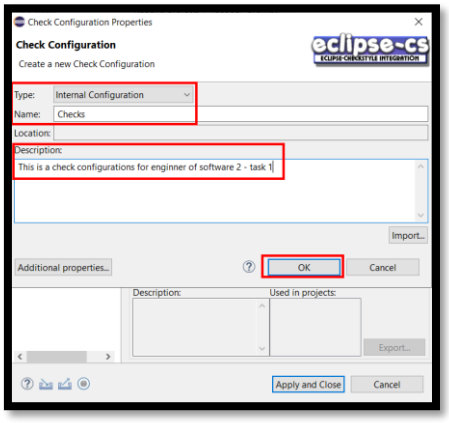
1. Go to “Window ◊ Preferences “and click “Checkstyle” on the side menu.



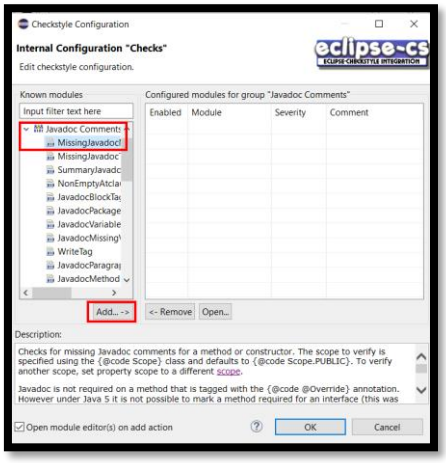
2. Check “Include rule names in violation messages” in General Settings
3. Click “New” to start creating a configuration file.



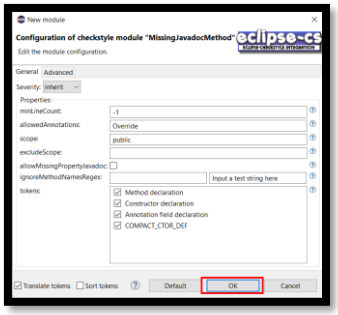
4. Select “Internal Configuration” under Type. For more information about the different types, refer to Chapter 21 Section 3 of the book [1].
5. For the Name, type “ Checks” and add a Description.
6. Click on “Ok” to create the file



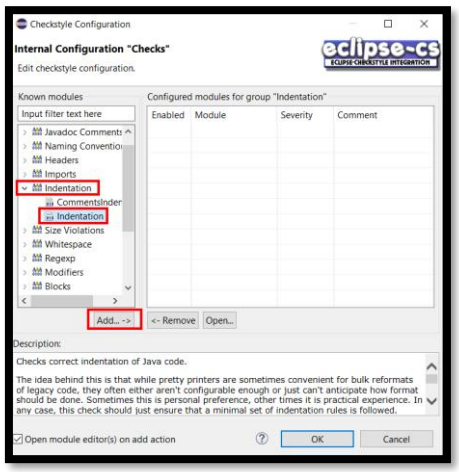
7. The configuration file will be ready to add new rules. Double click on it or click "Configure".
8. All available Checkstyle modules are displayed in groups and are ready to be added to the project. For more information on groups and modules, refer to Checkstyle [4]
9. Let's add some rules. Under "Javadoc 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".



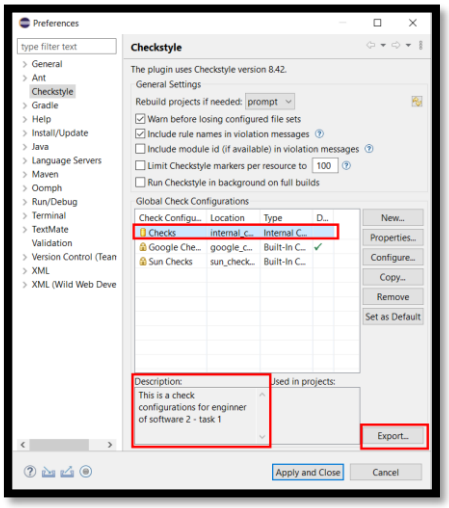
11. Now add a new rule for indentations. Under "Indentation", click "Indentation" then click "Add… ->"
12. Leave the default options, click "Ok".

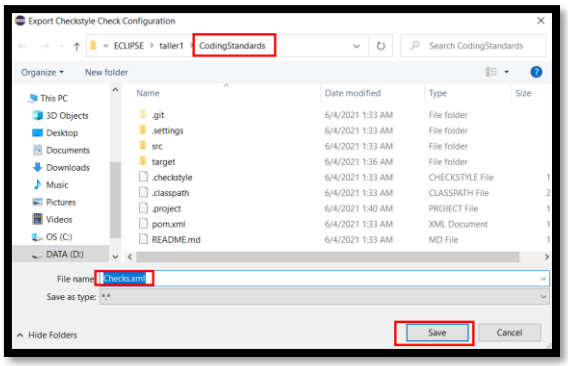


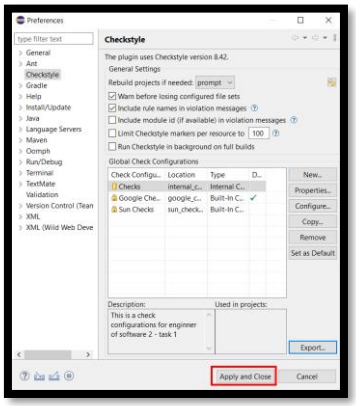
In case you add a wrong rule or misconfigure it, you can do the following:

- 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.

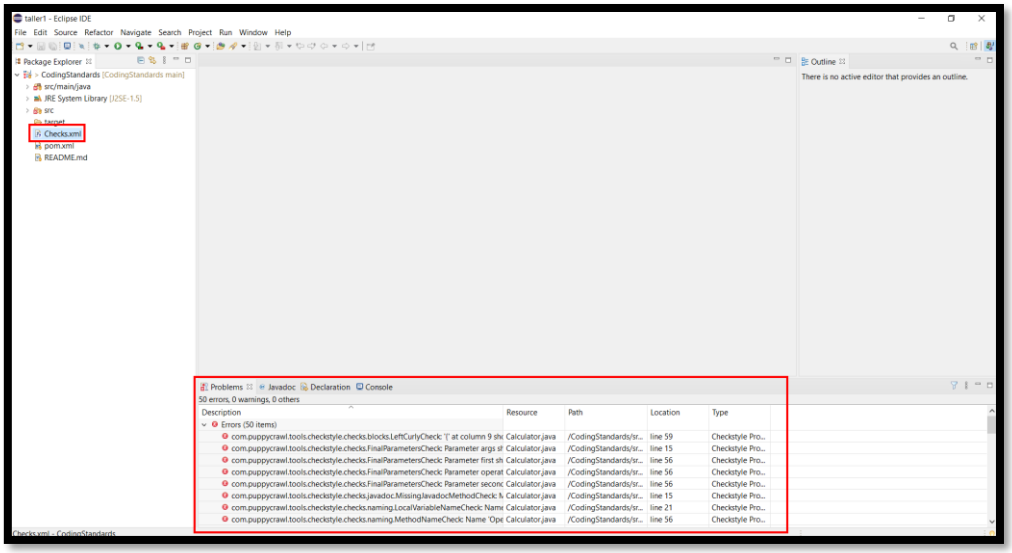
13. If you are happy with the results, click “Ok”
14. To export the file, select your check configuration in the Checkstyle configuration screen and click “Export” button and save it somewhere on your hard disk.





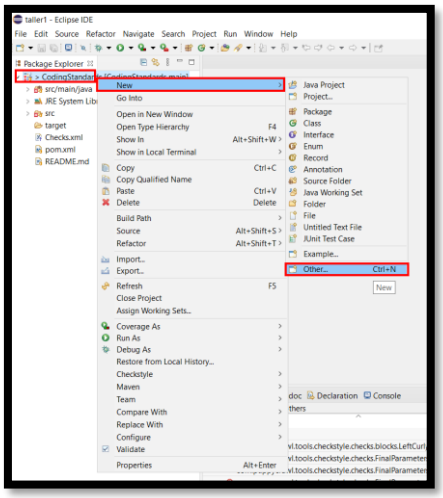


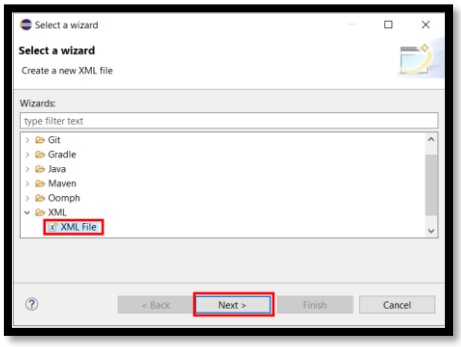
15. The configuration file is an XML file. Now you can publish it to your repository.
16. Check again the code and see the new warnings produced by Checkstyle.



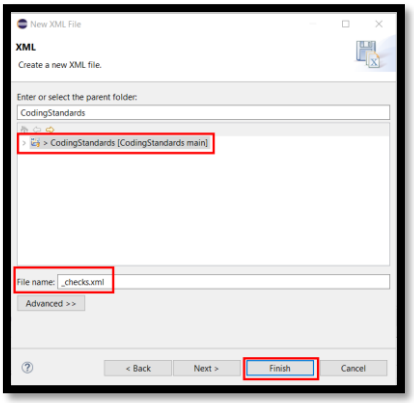
Part 5: Configure Checkstyle using the XML

1. Right click on the project, then “New >> Other >> XML >> XML File”.

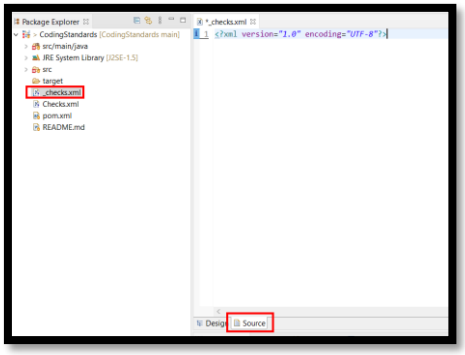




2. Select the project, enter a file name as “_checks”



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”:

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

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

```
<?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 name="LocalVariableName">
                        <property name="format" value="^[a-z](_?[a-zA-Z0-9]+)*$" />
                </module>
                <module name="MissingJavadocMethod ">
                        <property name="scope" value="private"/>
                        <property name="allowMissingPropertyJavadoc" value="true" />
                </module>
                <module name="JavadocType">
                        <property name="authorFormat" value="\S" />
                </module>
        </module>
</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>
```

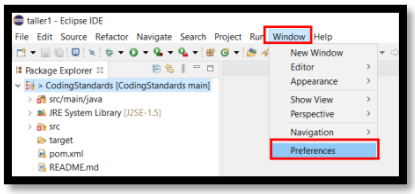
If the header needs to be changed in certain cases or a more sophisticated header is needed, please check Chapter 21 Section 5 of the book [1] for more details. The final file should look like this:

```
<?xml 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="scope" value="private"/>
                        <property name="allowMissingPropertyJavadoc" value="true" />
                </module>
                <module name="JavadocType">
                        <property name="authorFormat" value="\S" />
                </module>
        </module>
</module>
```

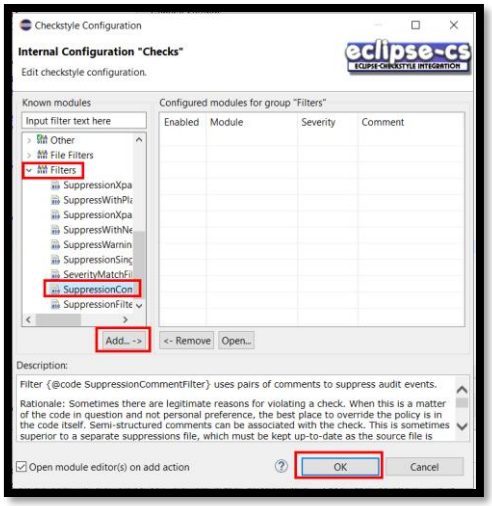
Part 7: Suppressing Checkstyle Tests

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.

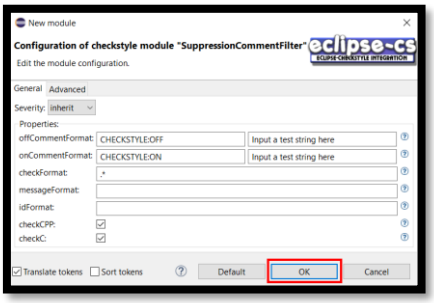
1. Go to "Window ◊ Preferences" and double-click your configuration file.



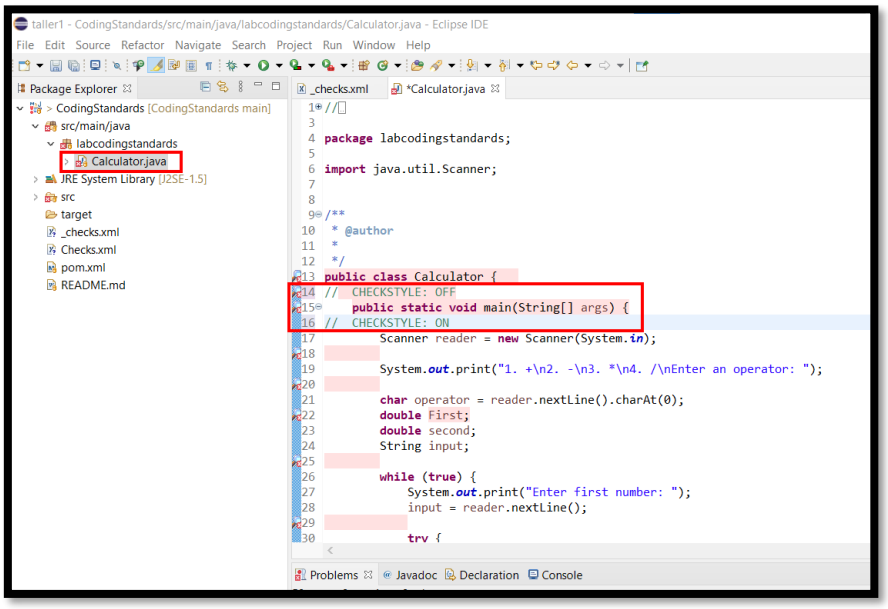
2. Search for "Suppression Comment Filter", select it and click "Add… ->"



3. Leave the default options, add the module and apply your changes



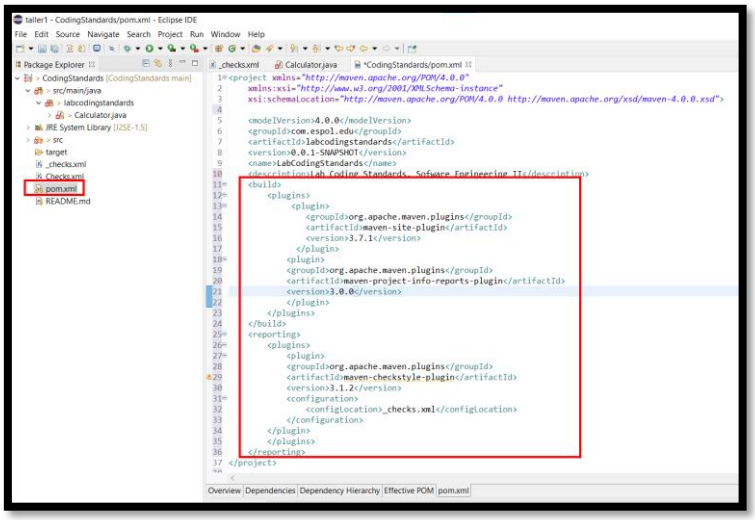
4. Go to the Calculator file and write "CHECKSTYLE: OFF" and "CHECKSTYLE: ON" around main method. If you remember, in Part 4, Step 11, we added a rule to verify indentation in all the files. The rule in that part of the code will be ignored in the report and no warning is shown. The other way to suppress 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.

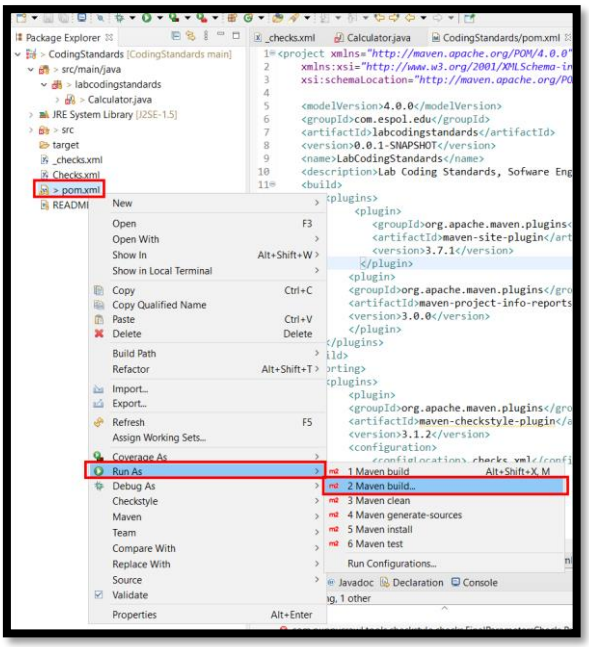
1. Go to the project tree and open “pom.xml"
2. Set up the basic configuration of your pom.xml. Add the following lines under project tag:



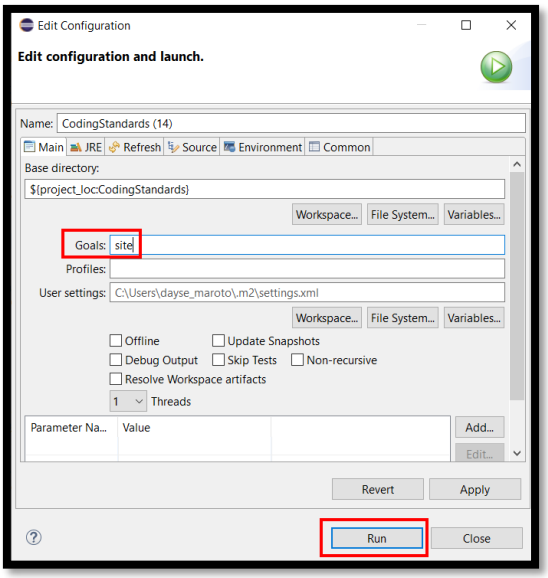
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

        <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>
        <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>
                <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-checkstyle-plugin</artifactId>
                    <version>3.1.2</version>
                        <configuration>
                            <configLocation>_checks.xml</configLocation>
                        </configuration>
                </plugin>
            </plugins>
        </reporting>
</project>
```

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.

1. Save the file, right Click on it, then “Run as >> Maven build…”

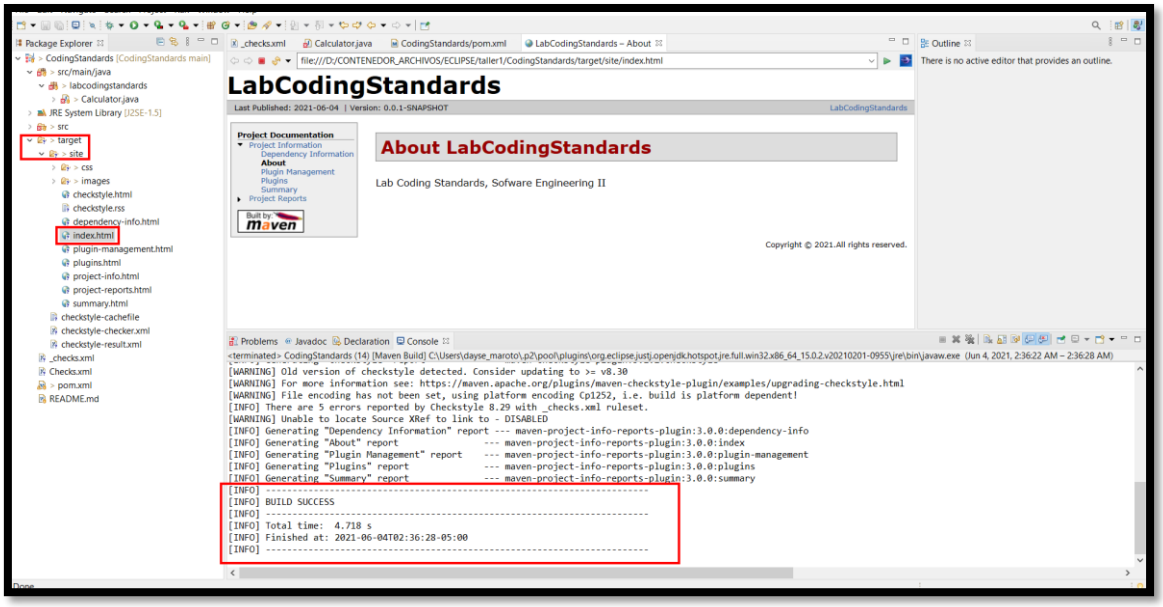


2. Type “site" for the Goals and Click Run.

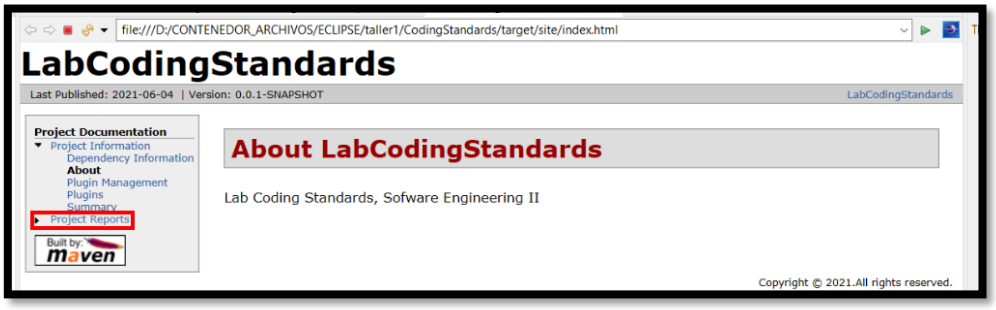


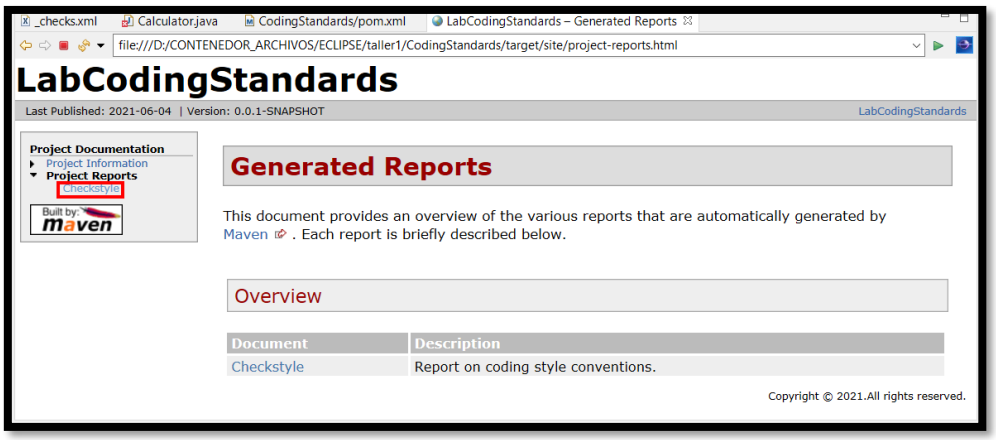
3. Wait for the process to finish.

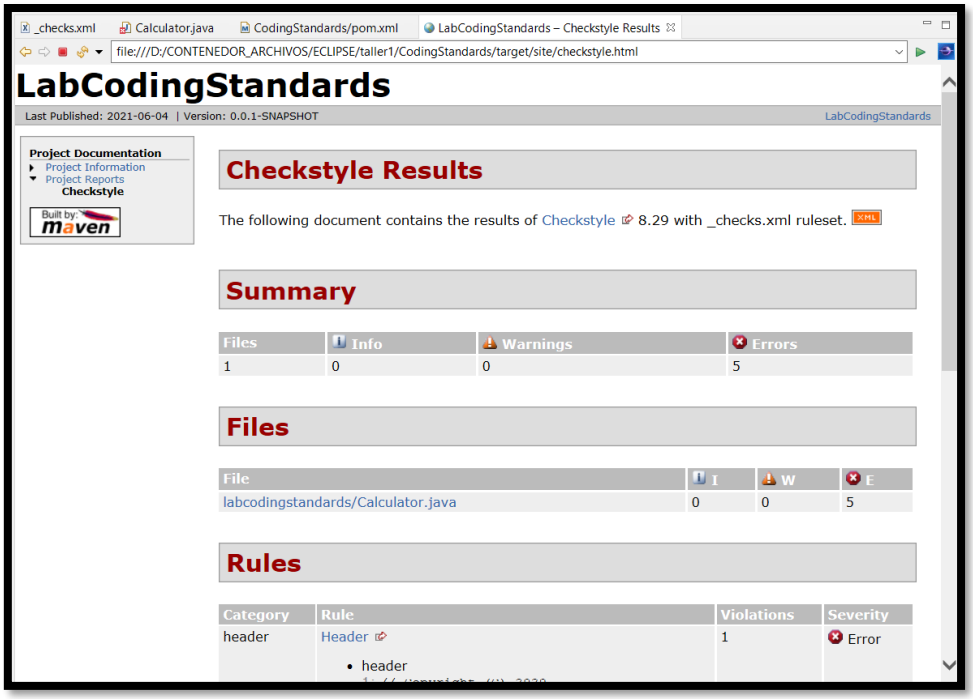
4. Go to your project directory, open “target >> site”
5. Several html files are show. Open “index.html".



6. Click on “Project Reports >> Checkstyle” to see a detailed Checkstyle report of your project.







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" • scope: "private"	2	Error
naming	LocalVariableName • format: "^[a-z](_?[a-zA-Z0-9]+)*$"	1	Error

Details

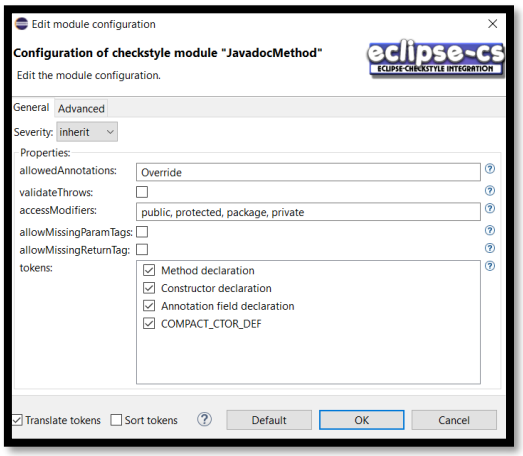
labcodingstandards/Calculator.java

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
Error	naming	LocalVariableName	Name 'First' must match pattern '^[a-z](_?[a-zA-Z0-9]+)*$'.	22
Error	javadoc	MissingJavadocMethod	Missing a Javadoc comment.	57

2) Challenge Task

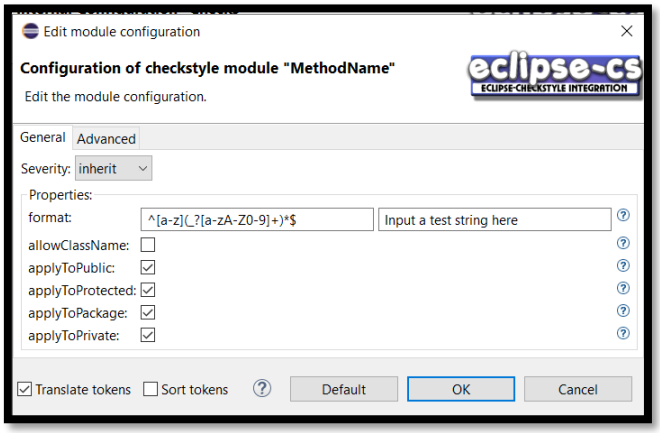
1. Add the following modules and properties to your XML configuration file:

- Javadoc Method.

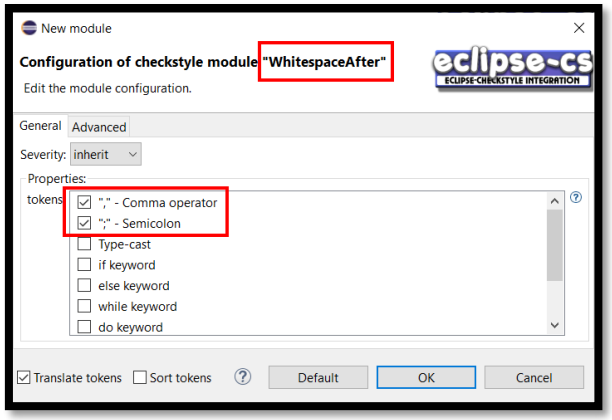


- Method Name, configure a property to match names that begin with a lower-case letter, followed by letters, digits, and underscores.

```
^[a-z](_?[a-zA-Z0-9]+)*$
```



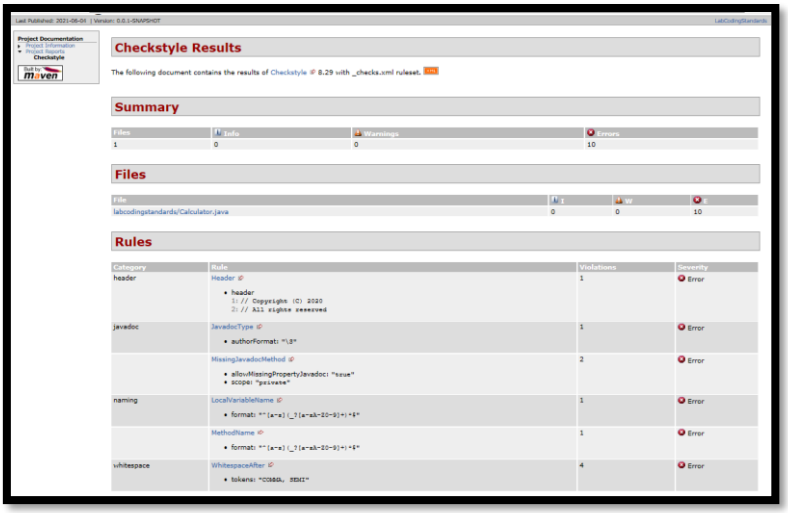
- Whitespace After, configure a property to check for whitespace only after COMMA and SEMI tokens.

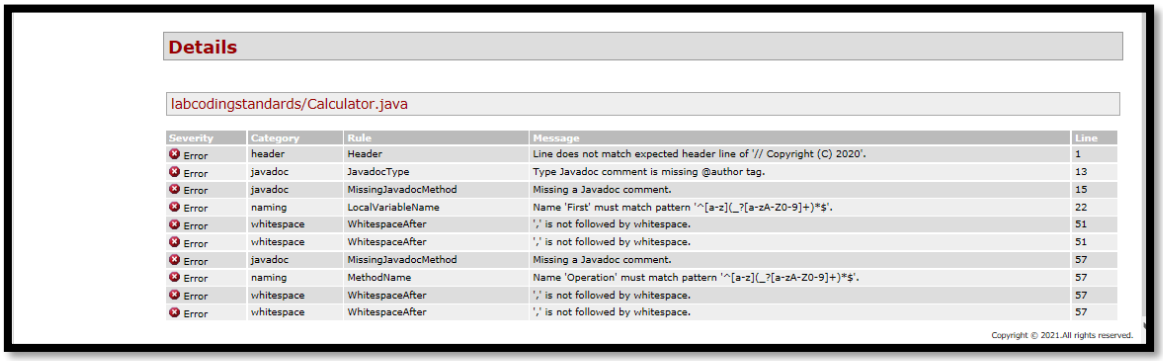


TIP: Refer to the Checkstyle website [4]

```
<?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]+)*$" />
                </module>
                <module name="MissingJavadocMethod ">
                        <property name="scope" value="private"/>
                        <property name="allowMissingPropertyJavadoc" value="true" />
                </module>
                <module name="JavadocType">
                        <property name="authorFormat" value="\S" />
                </module>
                <module name="JavadocMethod"/>
                <module name="MethodName">
                   <property name="format" value="^[a-z](_?[a-zA-Z0-9]+)*$"/>
                </module>

                <module name="WhitespaceAfter">
                  <property name="tokens" value="COMMA, SEMI"/>
                </module>
        </module>
</module>
```

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.

```
// Copyright (C) 2020
// All rights reserved

package labcodingstandards;

import java.util.Scanner;

/**
 * @author DayseMarotoLema
 *
 */
public class Calculator {
//CHECKSTYLE:OFF
/**
 * @param args
 */
    public static void main(final String[] args) {
//CHECKSTYLE:ON
        Scanner reader = new Scanner(System.in);

        System.out.print("1. +\n2. -\n3. *\n4. /\nEnter an operator: ");

        char operator = reader.nextLine().charAt(0);
        double first;
        double second;
        String input;

        while (true) {
```

```
            System.out.print("Enter first number: ");
            input = reader.nextLine();

            try {
    first = Integer.parseInt(input);
                break;
            } catch (NumberFormatException e) {
                System.out.println("Not valid!");
            }
        }

        while (true) {
            System.out.print("Enter second number: ");
            input = reader.nextLine();

            try {
                second = Integer.parseInt(input);
                break;
            } catch (NumberFormatException e) {
                System.out.println("Not valid!");
            }
        }

        Calculator cal = new Calculator();
        String result = cal.operation(first, second, operator);

        System.out.printf(result);
        reader.close();
    }

/**
 * @param f first
 * @param s second
 * @param op operation
 * @return result the result of the operation
 */
    private String operation(final double f, final double s, final char op) {
        double result = 0;
        switch (op) {
            case '1':
                result = f + s;
                break;
            case '2':
                result = f - s;
                break;
            case '3':
                result = f * s;
                break;
            case '4':
                result = f / s;
                break;
            default:
                return "Error! operator is not correct";
        }
        return "The result is: " + result;
    }
}
```

4. Generate a new report. It should not have any errors/warnings.