Assignment 10

Name: Varunkumar Pande

My-Mav: 1001722538

Part 1:

Manual analysis:

- Line 65 and 66 no input validation.

```
61      /* parse the HTTP request */
62      StringTokenizer st =
63          new StringTokenizer (request, " ");
64
65      command = st.nextToken();
66      pathname = st.nextToken();
67
68      if (command.equals("GET")) {
69          /* if the request is a GET
```

- In the run method for the try-catch block starting from line 102 the file reader (fr) is not closed in the catch block if an exception occurs.

```
101     /* try to open file specified by pathname */
102     try {
103         fr = new FileReader (pathname);
104         c = fr.read();
105     }
106     catch (Exception e) {
107         /* if the file is not found,return the
108            appropriate HTTP response code  */
109         osw.write ("HTTP/1.0 404 Not Found\n\n");
110         return;
111     }
112
113     /* if the requested file can be successfully opened
114        and read, then return an OK response code and
115        send the contents of the file */
```

- Variable ‘c’ is initialized as an integer but is assigned a string value post the file is being read, can lead to type errors.

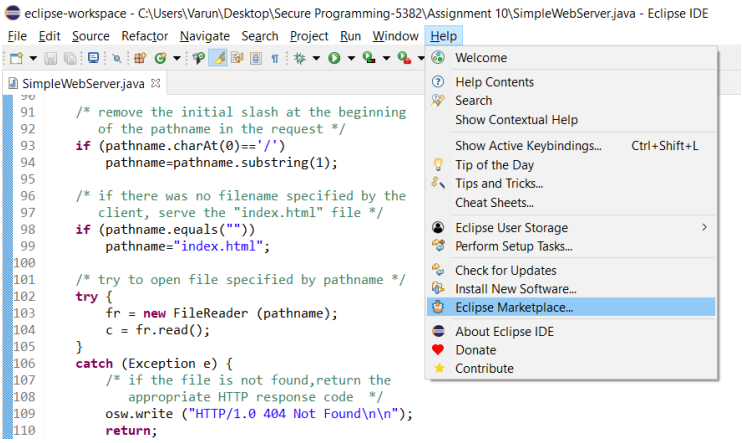
```
35      public void serveFile (OutputStreamWriter osw,
36                 String pathname) throws Exception {
37      FileReader fr=null;
38      int c=-1;
39      StringBuffer sb = new StringBuffer();
40
41      /* remove the initial slash at the beginning
42         of the pathname in the request */
43      if (pathname.charAt(0)=='/')
```

Tools Used:

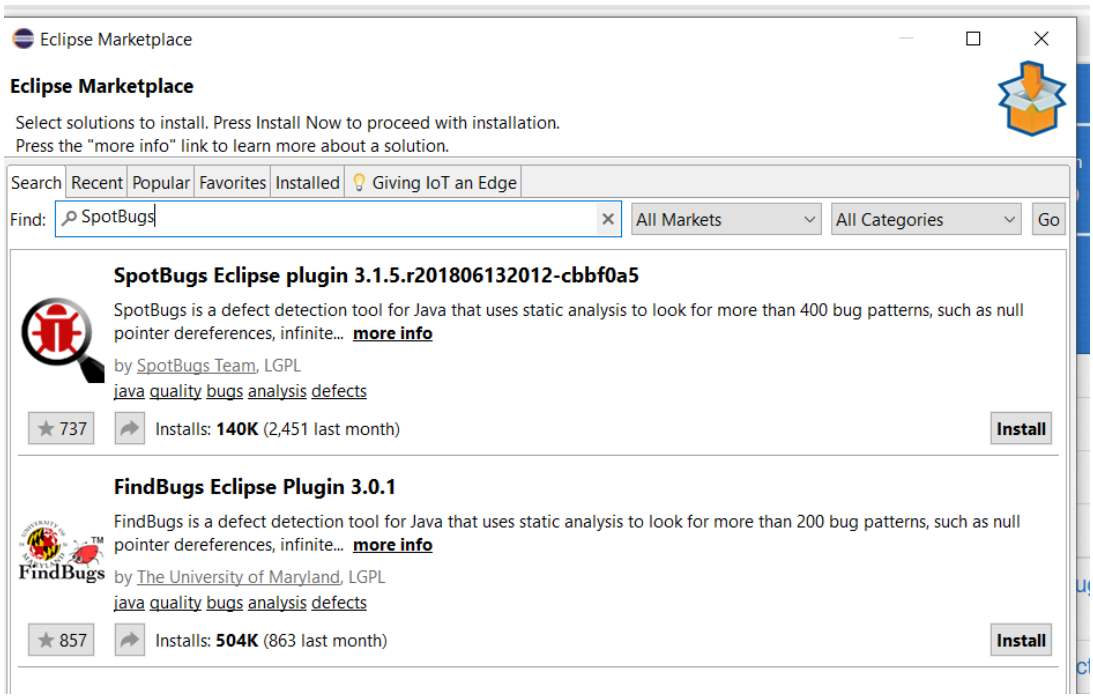
1. SpotBugs (version = 3.1.5.r201806132012-cbbf0a5)
2. SonarLint (version = 5.8.1)

SpotBugs installation procedure:

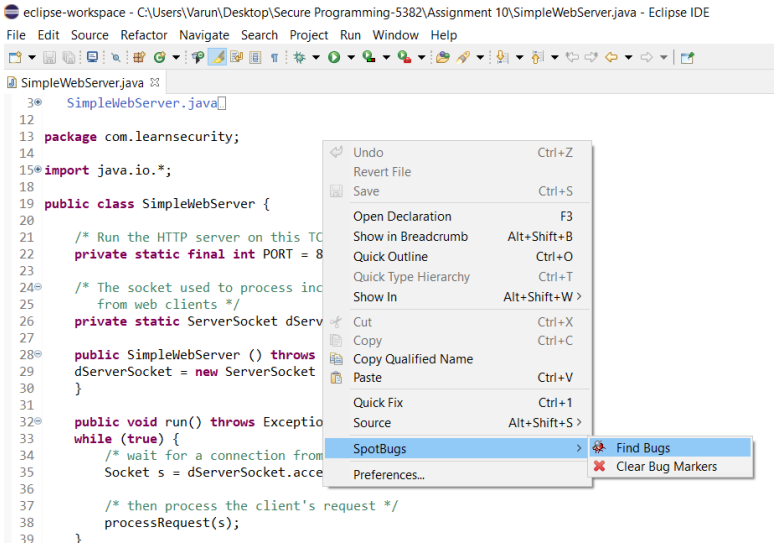
1. Go to Eclipse market place



2. Search and install SpotBugs.

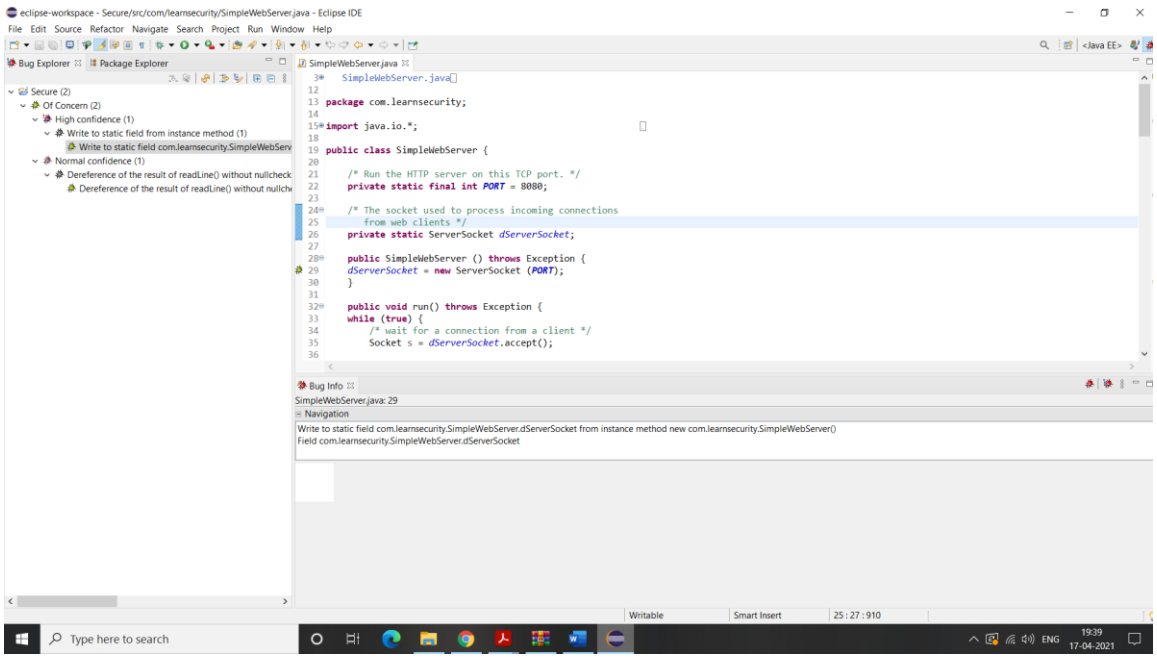


3. To run simply go to the java file that you want SpotBugs to analyse, right click and run SpotBugs.



Spotbugs Invocation Process:

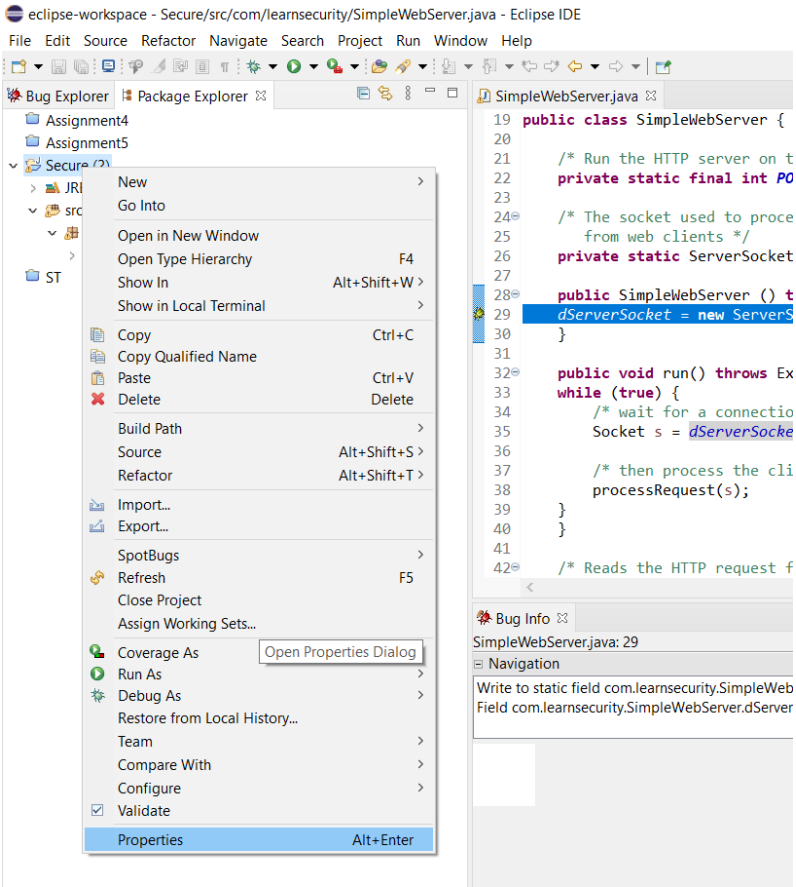
For the given Java file first we need to create a Java project and the package named “com.learnsecurity”. Run the SpotBugs tool, post which the Bug explorer window opens and display the bugs found by it as shown below.



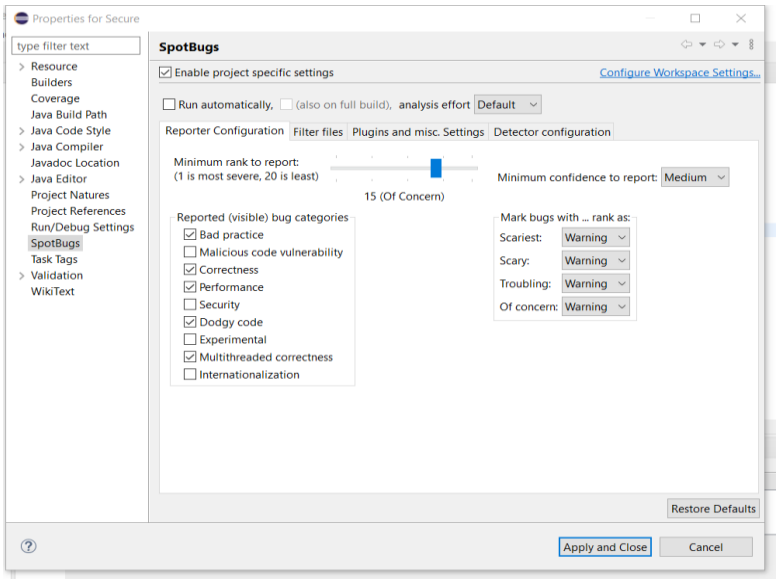
You can also click on each bug in the explorer to display the details around it, which gets displayed in the Bug Info section.

To change the settings related to the nature of bugs to be reported and confidence of tool in labelling them can be changed by going to the SpotBugs settings as shown below:

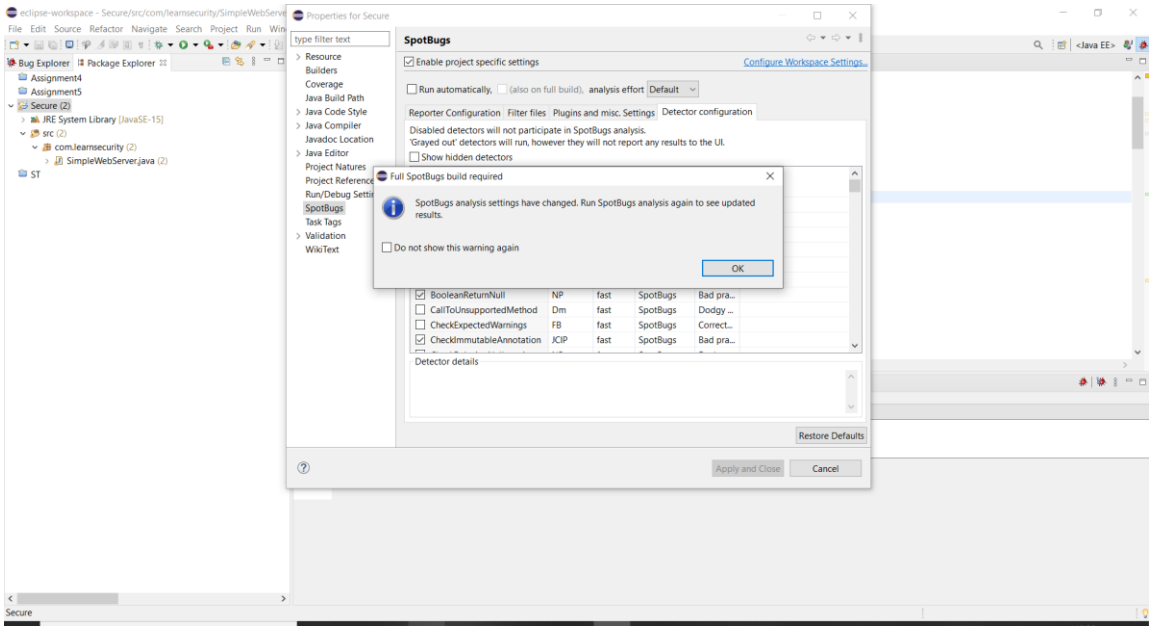
1. Go to project properties:



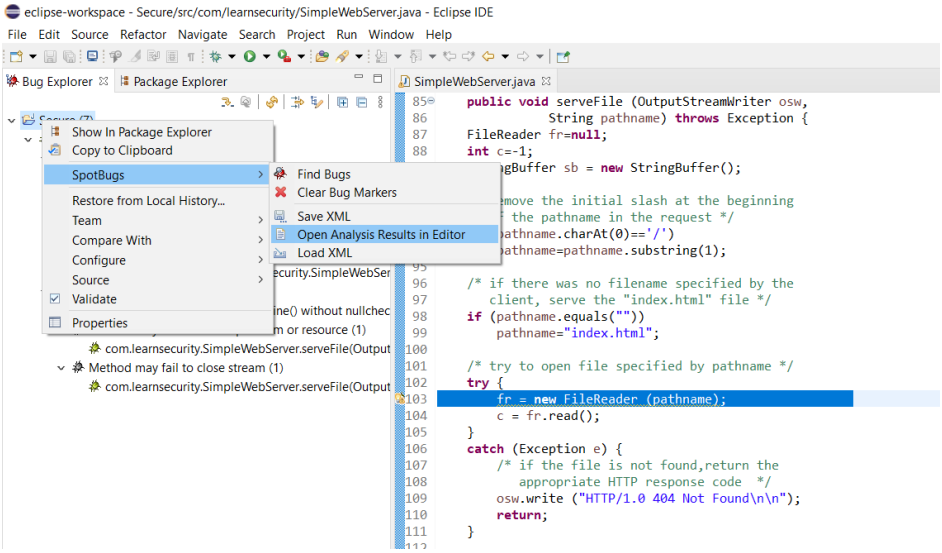
2. Select SpotBugs click on “enable project specific settings” and change as per required.

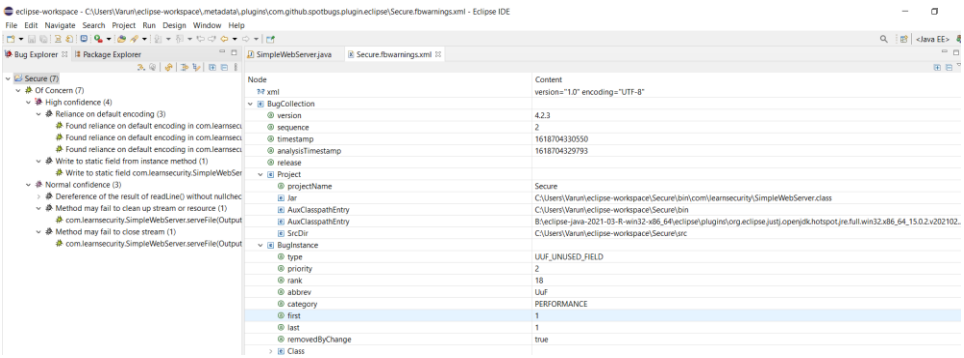


3. Once the settings are saved eclipse will ask to re-run SpotBugs in order to show new bugs that might appear because of the changed settings.



4. To save the view the analysis result you can either go to Analysis Results editor as shown below or you can view it in Bug explorer. As shown below you can even export the result in XML format.

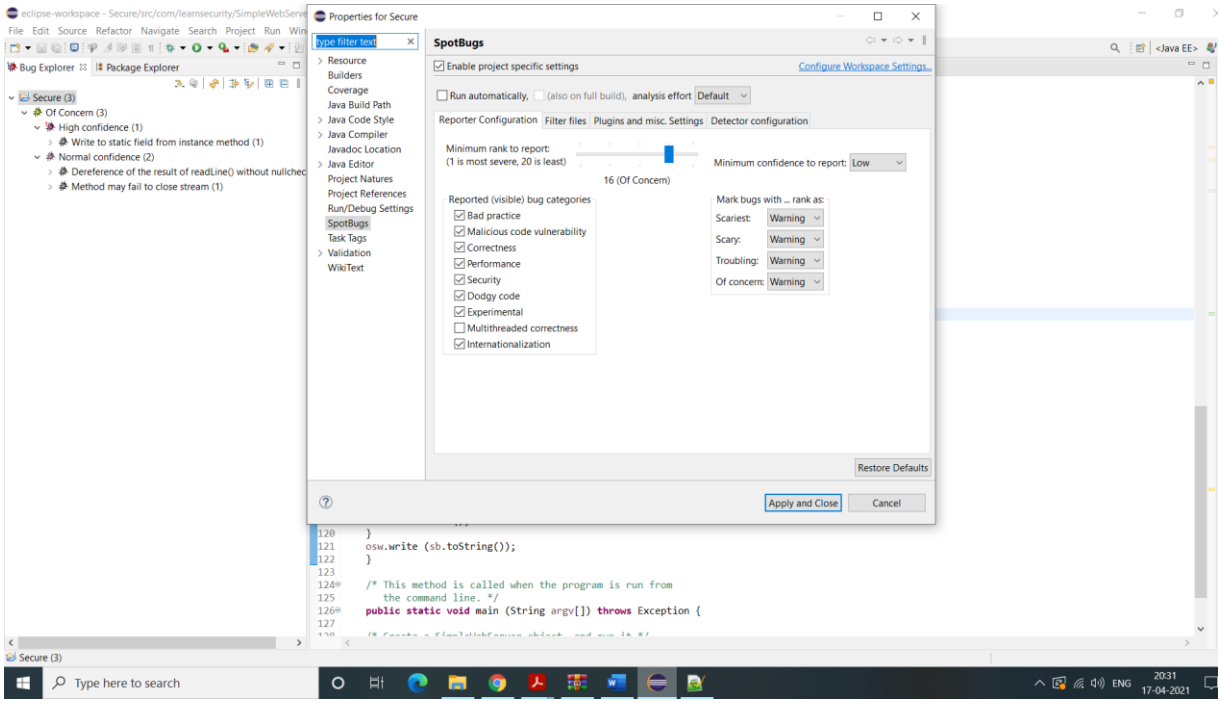




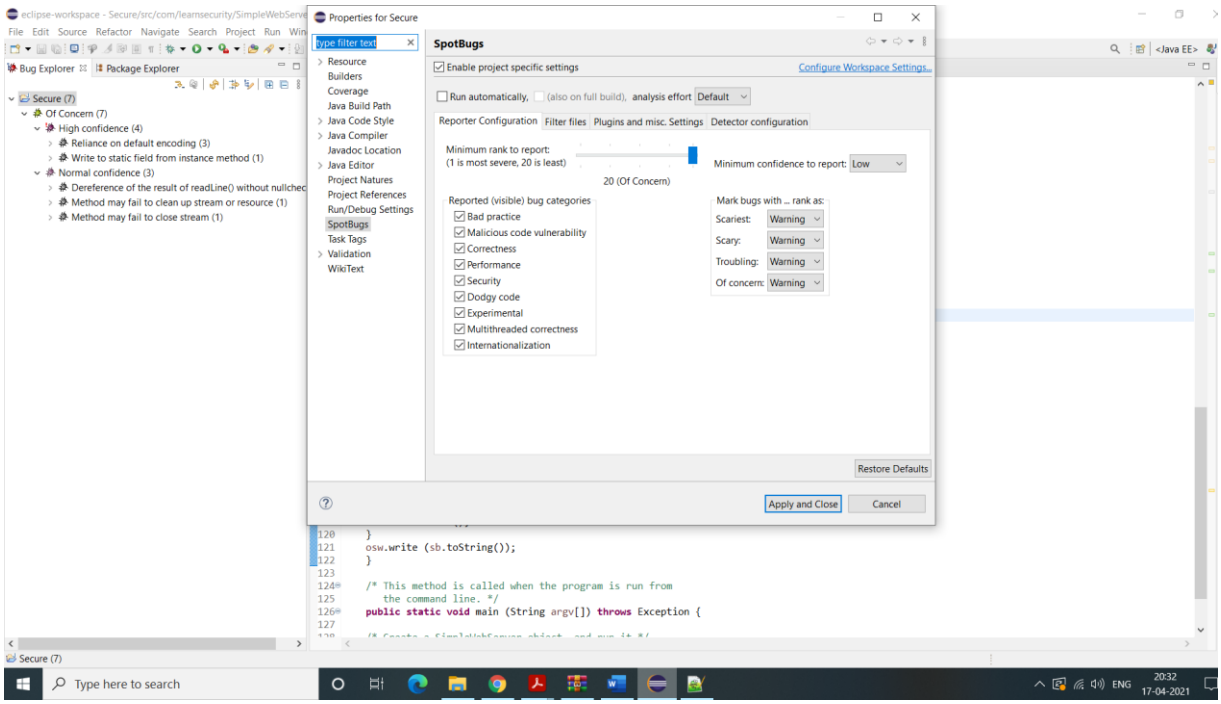
Experimenting with different bug settings and confidence levels:

The Bug explorer on the left shows the number of bugs and the properties page shows the current configuration:

1.

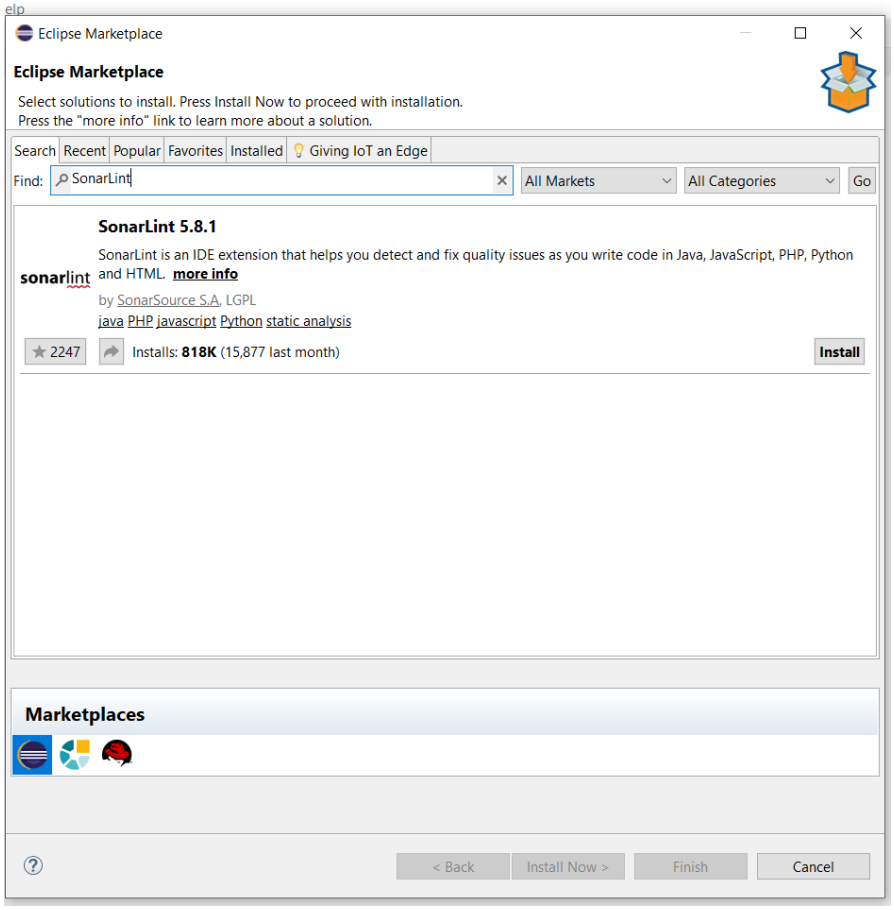


2.

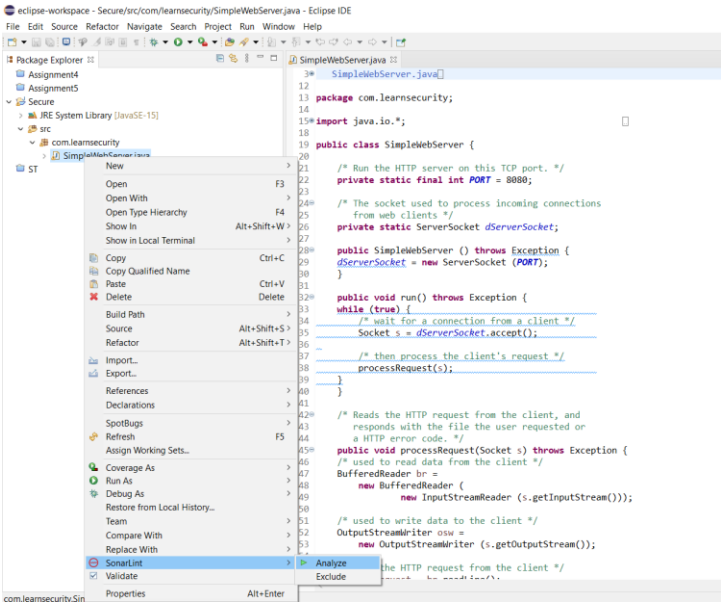


SonarLint installation procedure:

1. Go to Eclipse market place.
2. Search and install SonarLint.

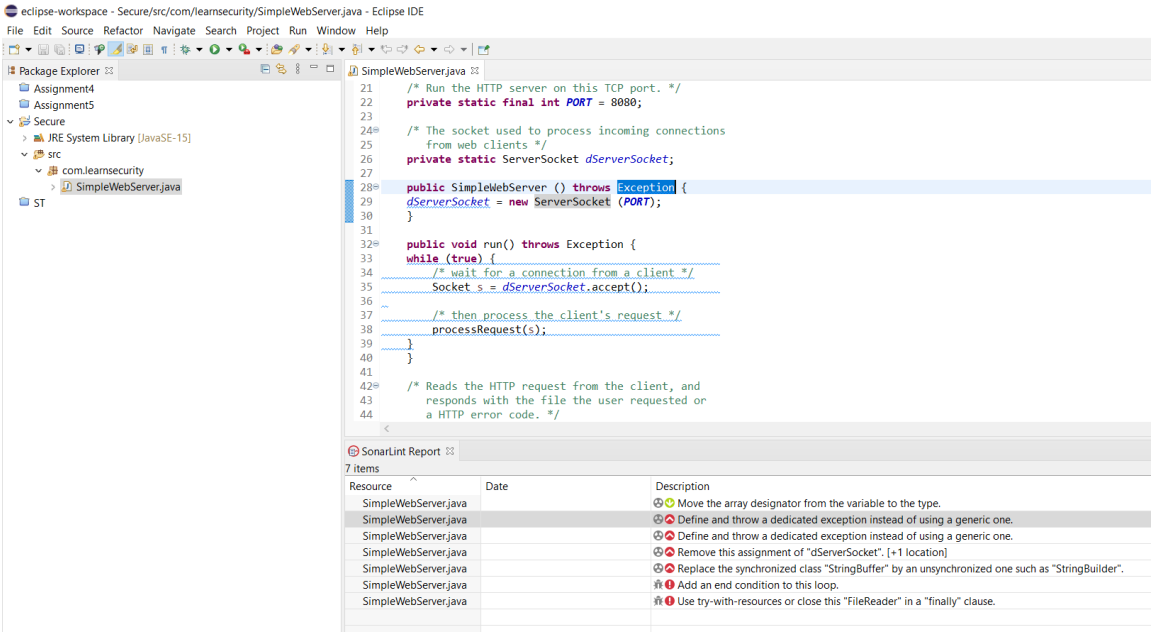


3. To Run SonarLint, we simply select the java file and run the SonarLint analyses as shown below.

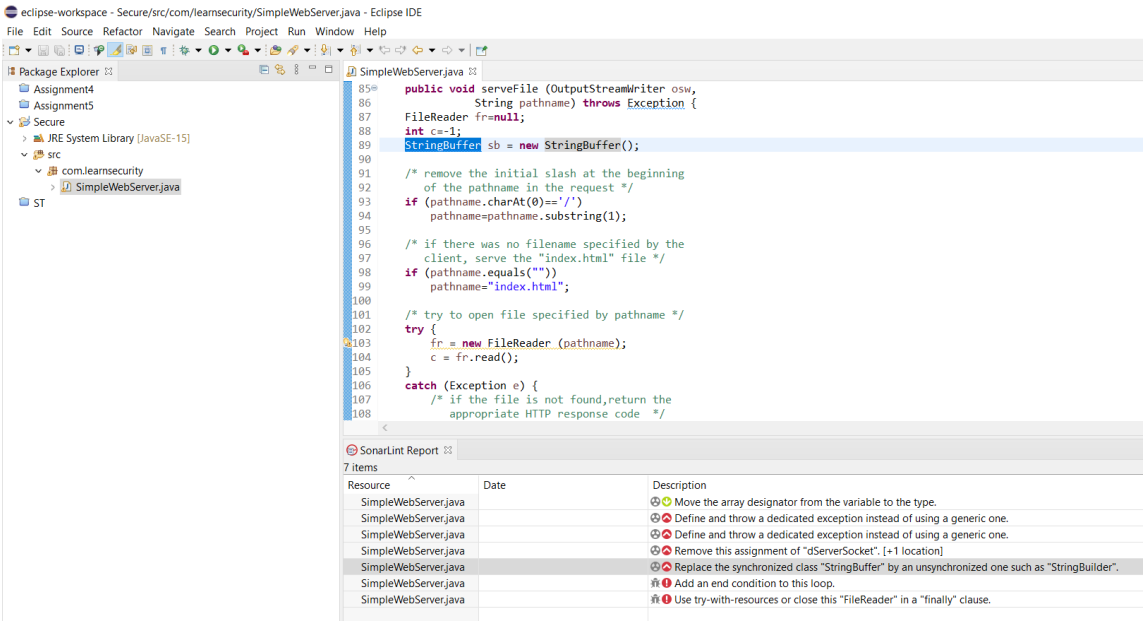


SonarLint Invocation process:

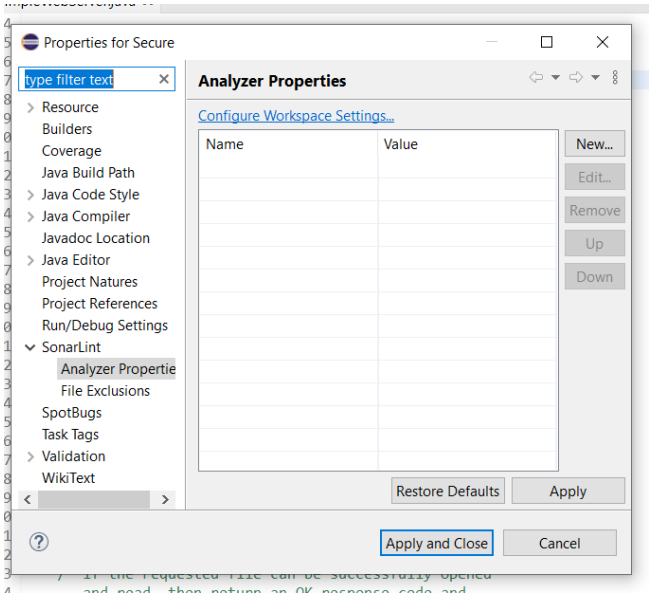
1. Since we already have created a project while using the prior tool, now we can simply run SonarLint analyser directly by right clicking on the java file in the project explorer and get the bugs as shown below:



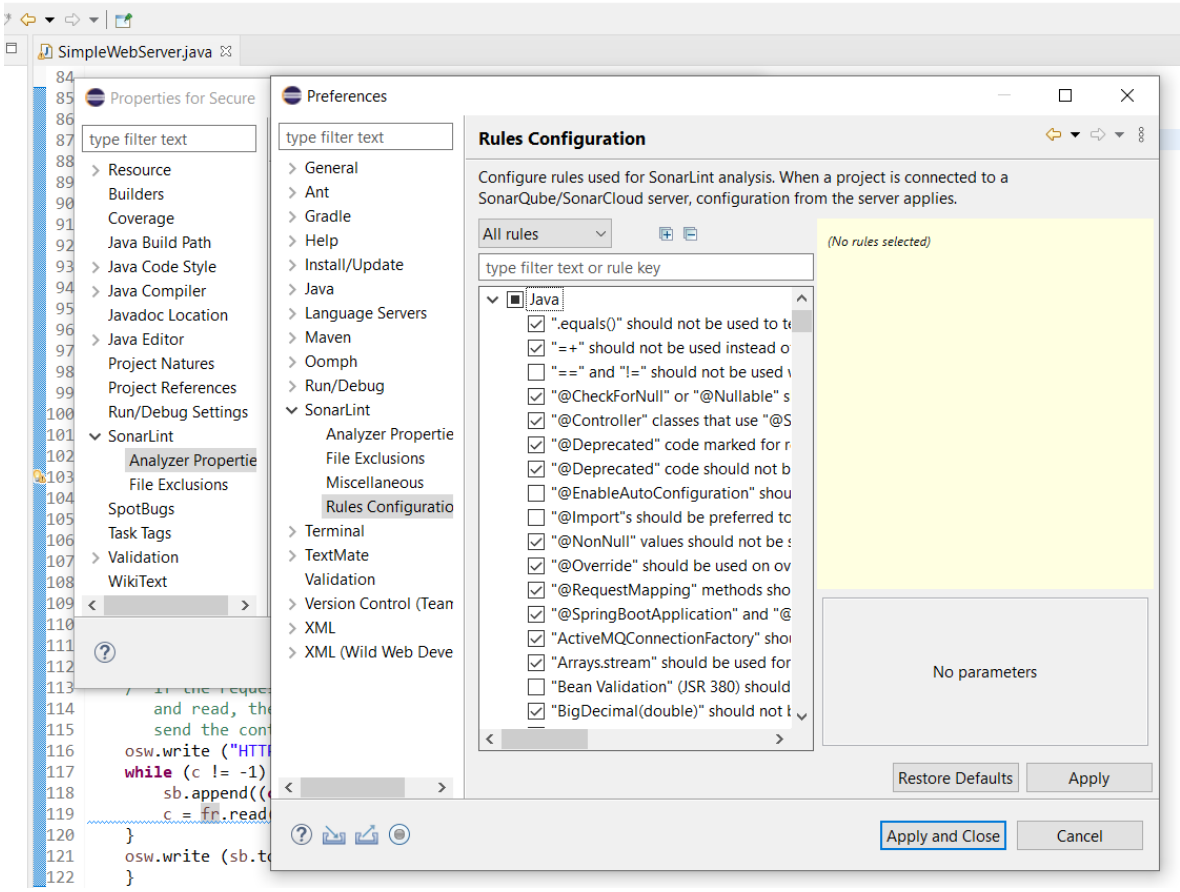
2. By clicking the rows under SonarLint Report tab we can directly get to the line in code that has the bug.



To change the settings related to analyser we can configure new rules by entering them under the “Analyzer properties” tab as shown below:

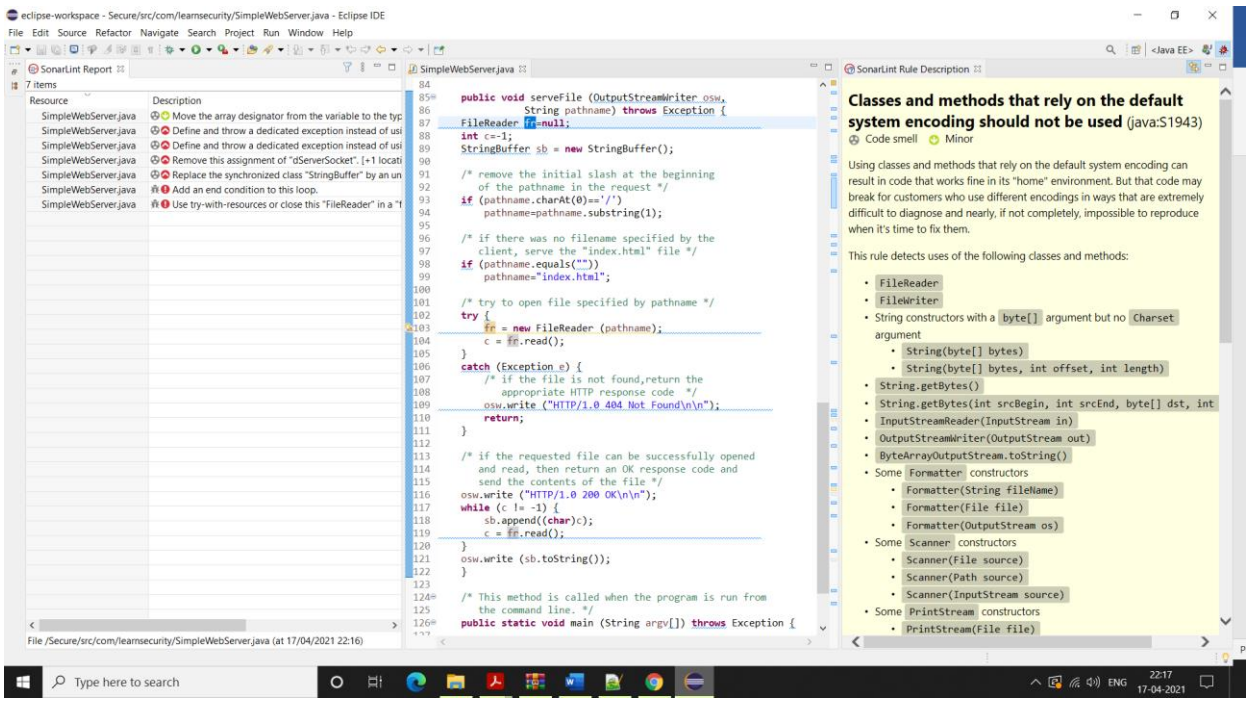


We can also change the rule configurations that are used to analyse the file by going into the rules configuration page and editing the settings as required.

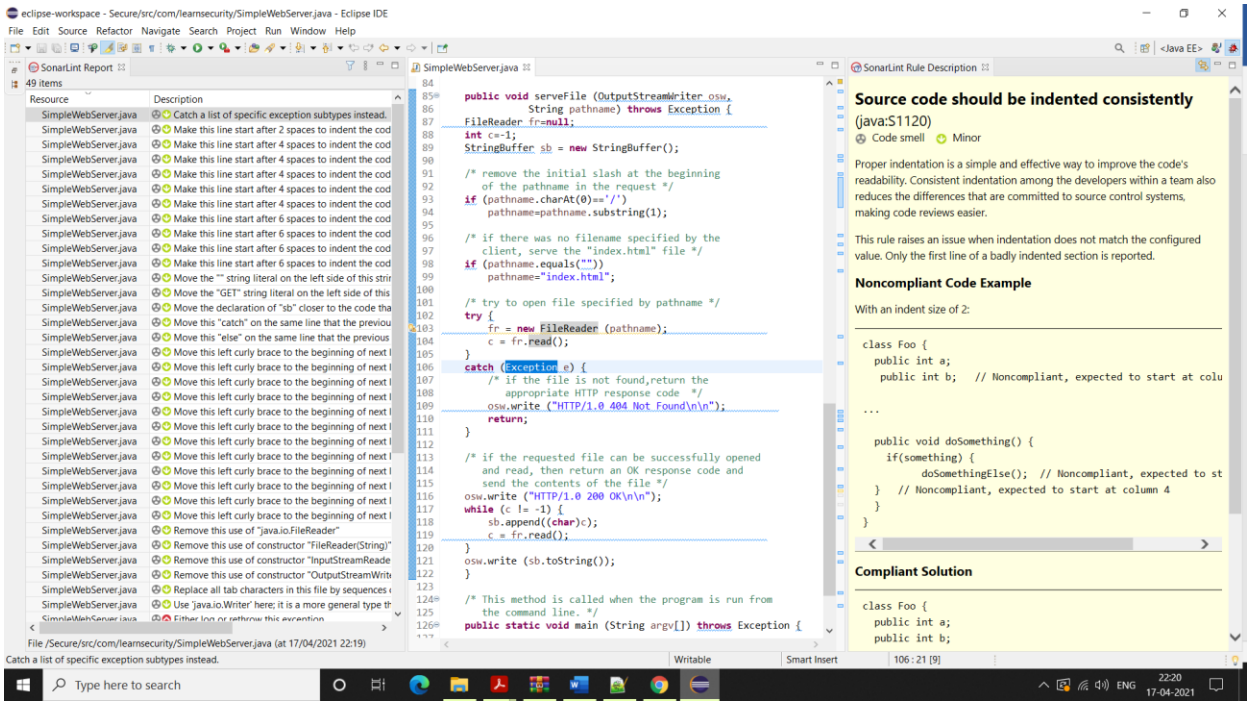


Experimenting with different rules selected in the “Rules configuration”:

1. Selecting a few rules.



2. Selecting all the possible rules.



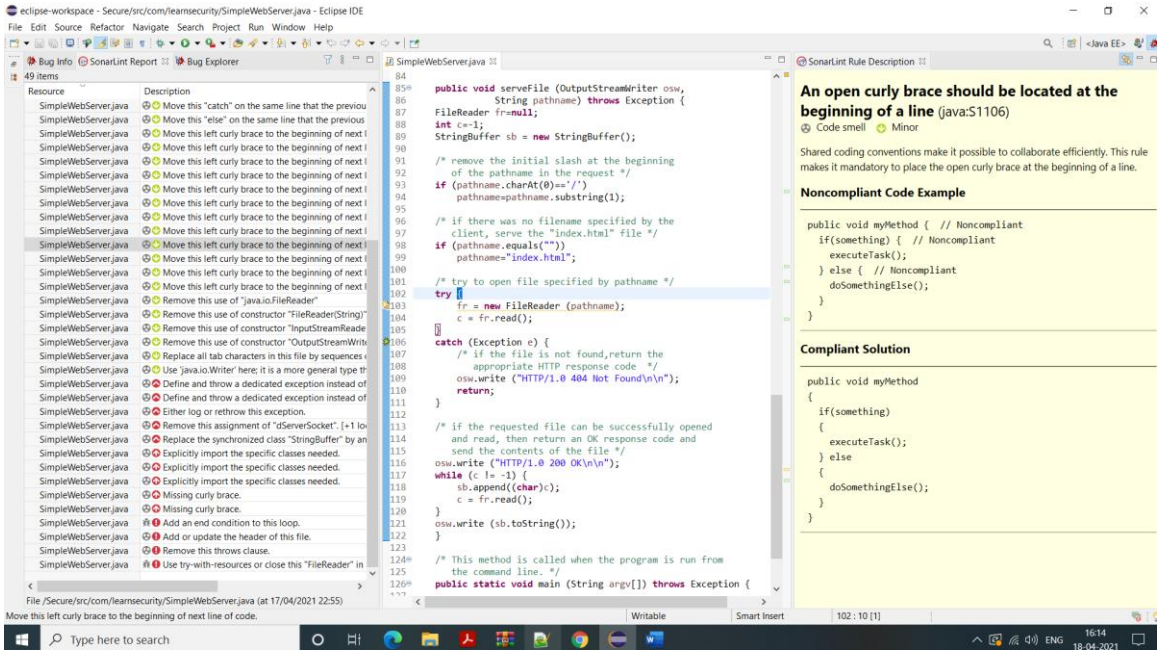
The SonarLint Rule Description tab helps to get more details on the bugs that are identified by it. It also provides few examples to better explain the code.

Comparison/Contrast Tools:

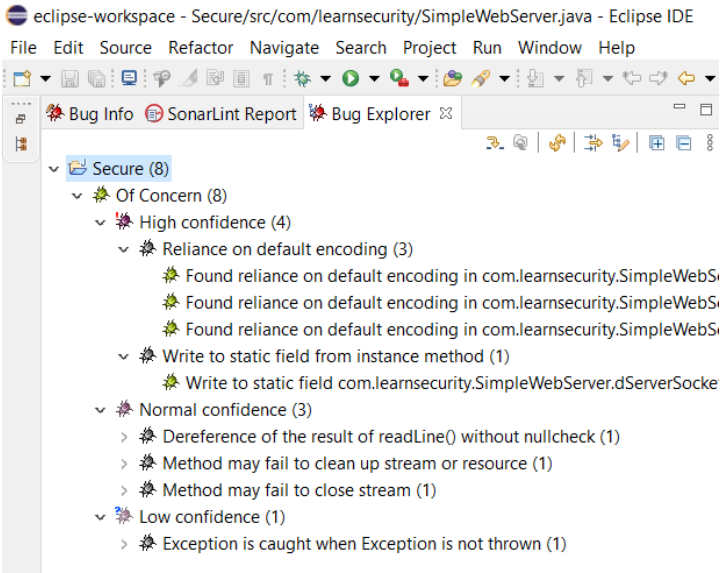
SonarLint	SpotBugs
SonarLint analyses the source code.	SpotBugs uses BCEL to analyze Java bytecode.
SonarLint can fit into many categories of tool like Style checking, Bug finding, Security review.	SpotBugs focuses on Bug finding process using Type and Property checking.
Found it tough to export the results, need to use third party plugins or link with SonarQube to extract the results.	Easy XML export available. Makes it easier to report the bugs on developer dashboards.

Show an example (if one exists) of a finding that is reported by one tool and not others.

SonarLint reports errors relating to code formatting also but SpotBugs doesn't highlight those issues:

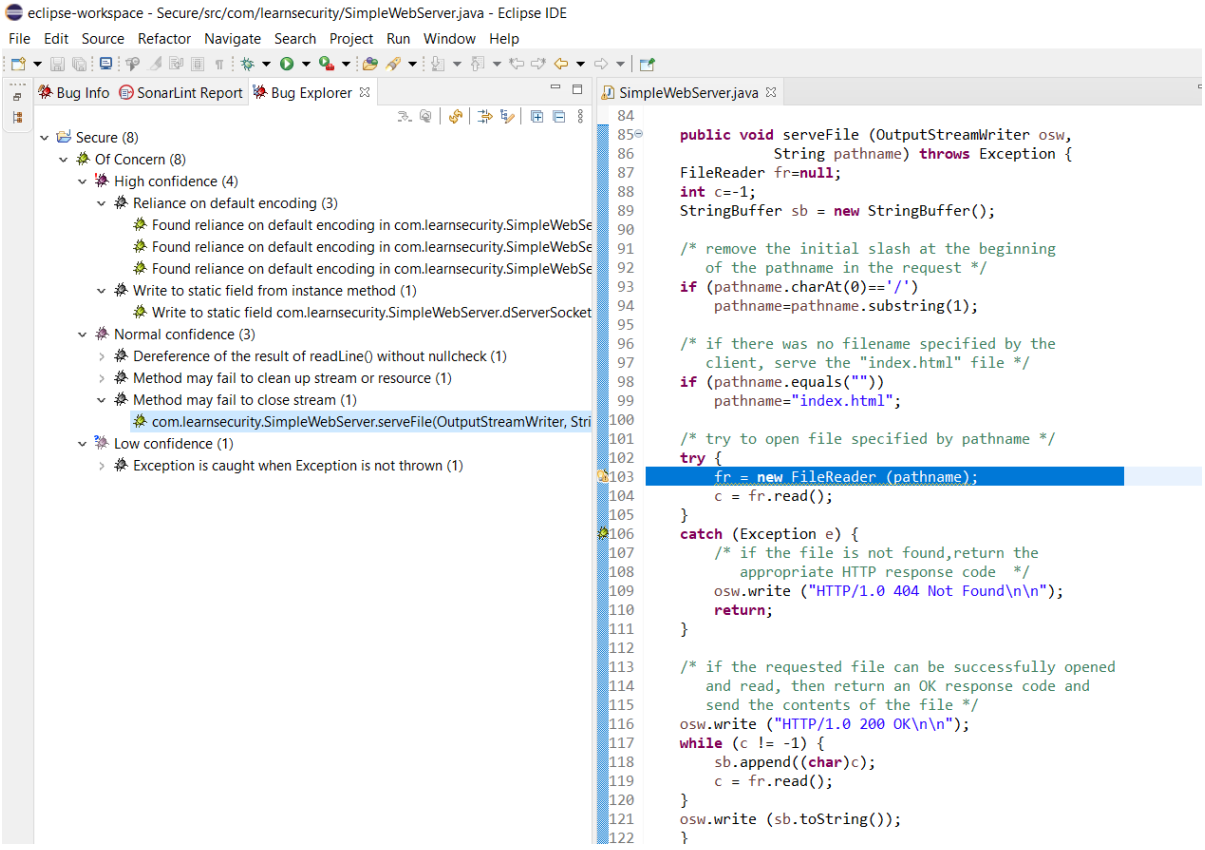


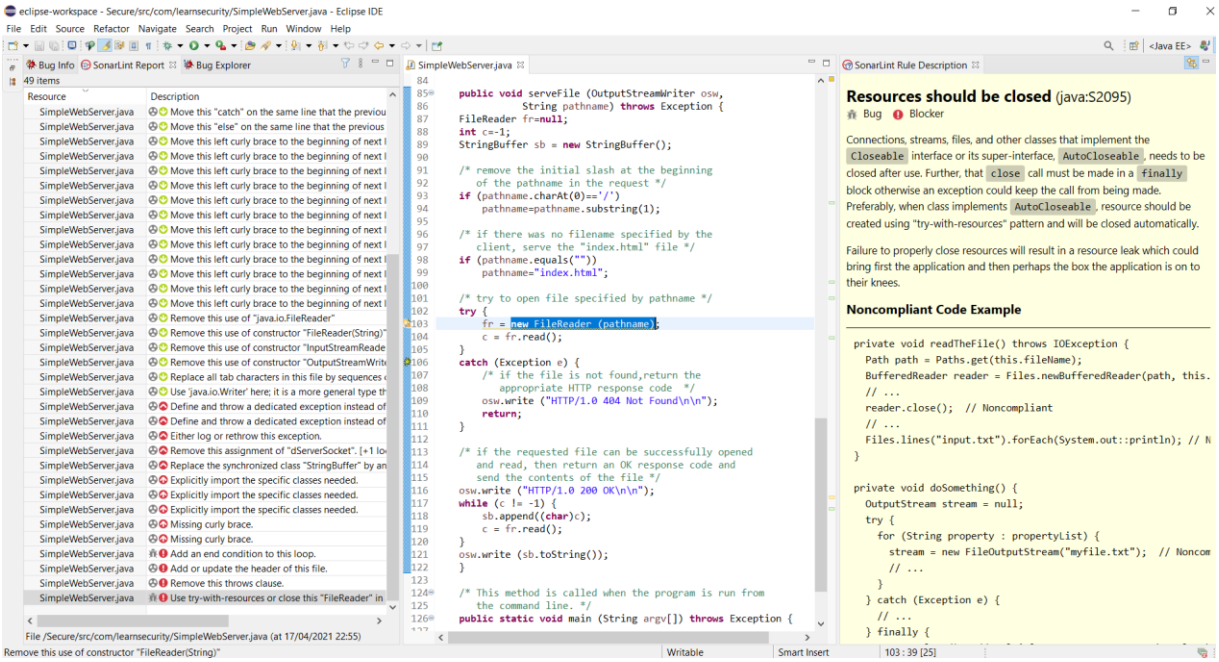
The below screenshot shows SpotBugs output for the same java code.



Show an example (if one exists) of a finding reported by multiple tools.

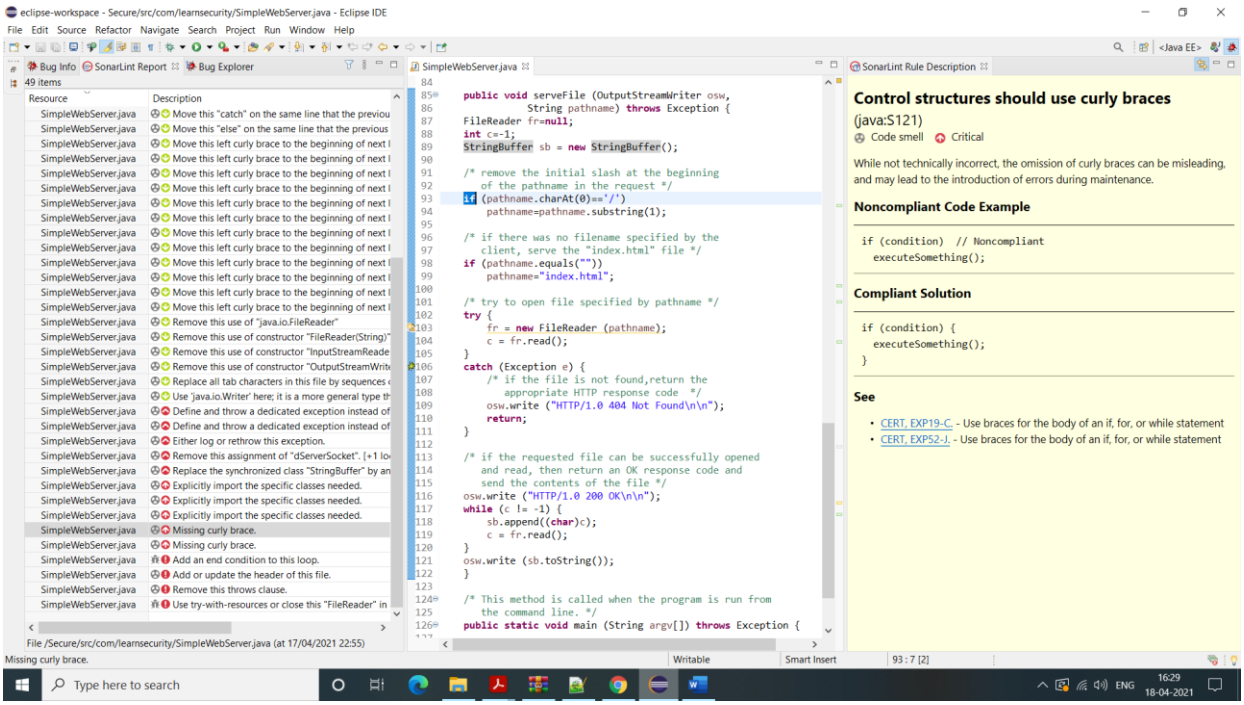
One of the errors that was reported in both of the tools was relating to the possibility of “file reader” not being closed if an exception is caused (i.e., resource leaking).





For the known flaw in the code used, document which tools reported it (true negative) and which tools did not (false positive).

One of the known flaws was relating to the “file reader” not being closed if an exception is caused, was caught by both of the tools. But SonarLint reported few false positive related to styling, like in the following case since “if-condition” is a single statement we do not require braces for it.



Part 2:

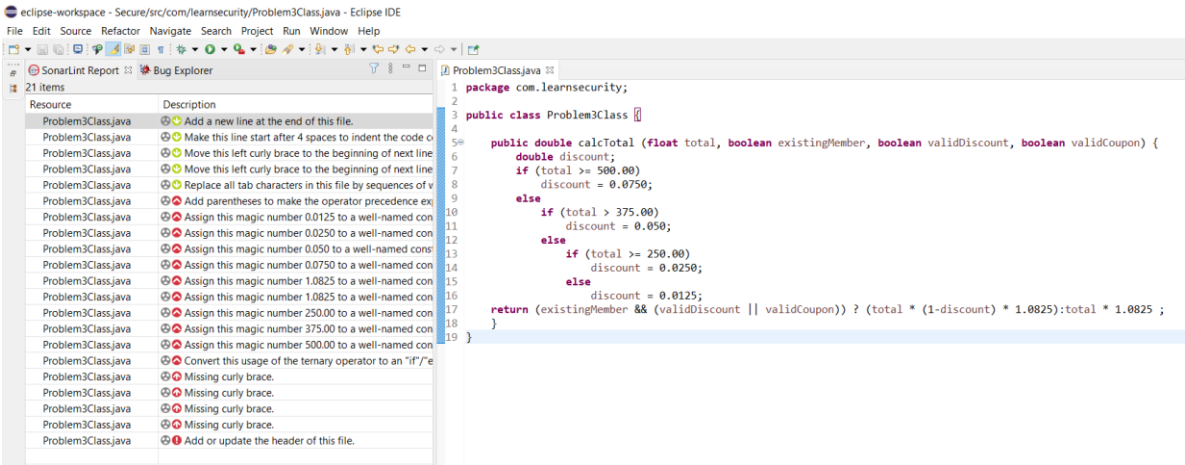
My Code:

```
package com.learnsecurity;

public class Problem3Class {

	public double calcTotal (float total, boolean existingMember, boolean
validDiscount, boolean validCoupon) {
		double discount;
		if (total >= 500.00)
			discount = 0.0750;
		else
			if (total > 375.00)
				discount = 0.050;
			else
				if (total >= 250.00)
					discount = 0.0250;
				else
					discount = 0.0125;
	return (existingMember && (validDiscount || validCoupon)) ? (total * (1-
discount) * 1.0825):total * 1.0825 ;
	}
}
```

Manual Analysis:

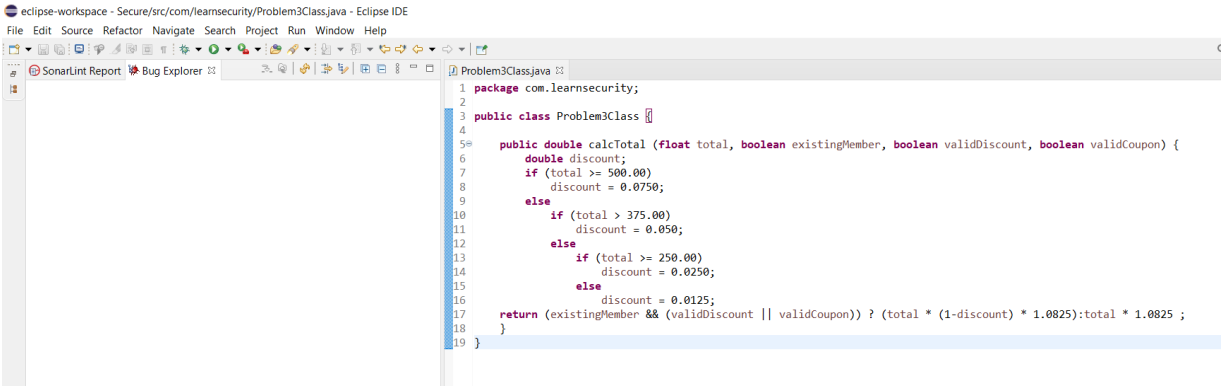
The above code does not contain any prominent bugs except for the fact that the total is assigned a float data type which can lead to potential problems due to overflow of values.

SonarLint Analysis:



SonarLint did complain about issues relating to styling of code like missing parenthesis for if-statements and using if-else instead of ternary comparison operator. But those can be ignored because the if clause is just assigning values to variable and use of ternary operator makes the code look shorter.

SpotBugs did not report any issues in the code, as shown in the below screenshot:



Fixes:

Since the suggested errors were only related to code styling and not Bugs, there is no necessity to implement the changes suggested for the outlined issues.