Assignment 1

Name: Charlie Kuhl Student number: 46598065

# Sonarlint Analysis

## A.1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S-Number** | **Description** | **Type** | **Severity** | **Number of Violations** |
| S106 | Standard outputs should not be used directly to log anything | Code smell | Major | 22 |
| S120 | Package names should comply with a naming convention | Code smell | Minor | 6 |
| S2479 | Whitespace and control characters in literals should be explicit | Code smell | Critical | 4 |
| S125 | Sections of code should not be commented out | Code smell | Major | 4 |
| S1186 | Methods should not be empty | Code smell | Critical | 2 |
| S1066 | Collapsible "if" statements should be merged | Code smell | Major | 2 |
| S2589 | Boolean expressions should not be gratuitous | Code smell | Major | 2 |
| S2699 | Tests should include assertions | Code smell | Blocker | 1 |
| S2189 | Loops should not be infinite | Bug | Blocker | 1 |
| S1948 | Fields in a "Serializable" class should either be transient or serializable | Code smell | Critical | 1 |
| S3252 | "static" base class members should not be accessed via derived types | Code smell | Critical | 1 |
| S3776 | Cognitive Complexity of methods should not be too high | Code smell | Critical | 1 |
| S1161 | "@Override" should be used on overriding and implementing methods | Code smell | Major | 1 |
| S2142 | "InterruptedException" should not be ignored | Bug | Major | 1 |
| S4973 | Strings and Boxed types should be compared using "equals()" | Bug | Major | 1 |
| S1126 | Return of boolean expressions should not be wrapped into an "if-then-else" statement | Code smell | Minor | 1 |
| S1130 | "throws" declarations should not be superfluous | Code smell | Minor | 1 |
| S1612 | Lambdas should be replaced with method references | Code smell | Minor | 1 |
| S1643 | Strings should not be concatenated using '+' in a loop | Code smell | Minor | 1 |
| S2160 | Subclasses that add fields should override "equals" | Code smell | Minor | 1 |
| S2272 | "Iterator.next()" methods should throw "NoSuchElementException" | Bug | Minor | 1 |

## A.1.2

**S106 - Standard outputs should not be used directly to log anything**

This warning has the highest frequency out of all warnings while also having major severity.

Text

Description automatically generated

Loggers should be used in place of the System Output Stream as this allows for better monitoring of the program. Loggers provide formatting and structure to messages, include important information such as timestamps and categorization, and allows outputting to a file for later analysis. Loggers also prevent messages containing internal information from being returned to end-users and potential attackers.

**S2479 - Whitespace and control characters in literals should be explicit**

This has the highest frequency out of all critical warnings.

Text

Description automatically generated

The affected code fragments are located in the SQL connection manager and can be found in strings that are used to hold multi-line SQL queries. SQL queries require specific formatting, which the code attempts to follow by using newline characters and whitespace at the start of the two middle strings. Since the whitespace is not using control characters it can be difficult to determine if it is supposed to be a single space or a tab. This can cause bugs when the string is interpreted incorrectly.

In this case the problem would be very simple to fix since all you need to do is replace the whitespace with a “\t” (TAB) control character.

**S2699 - Tests should include assertions**

While this warning only appears once, it has the highest severity of “blocker” and thus has a high probability to negatively impact the application.

Text

Description automatically generated

This code fragment is from AppTest.java which should hold the test cases for the application to ensure that the application is functioning as intended. Junit assertions has been imported but no assertions have been used in the test case, effectively creating a test that does not test anything or throw any exceptions.

## A.1.3

**S106 - Standard outputs should not be used directly to log anything**

**S2479 - Whitespace and control characters in literals should be explicit**

**S2189 - Loops should not be infinite**

# PMD Analysis

## A.1.4

|  |  |  |  |
| --- | --- | --- | --- |
| **PMD rule** | **Priority** | **Category** | **Number of Violations** |
| UncommentedEmptyMethodBody | Medium (3) | Documentation | 3 |
| CloseResource | Medium (3) | Error prone | 2 |
| LiteralsFirstInComparisons | Medium (3) | Best practices | 2 |
| ClassWithOnlyPrivateConstructorsShouldBeFinal | High (1) | Design | 1 |
| ImplicitSwitchFallThrough | Medium (3) | Error prone | 1 |
| CompareObjectsWithEquals | Medium (3) | Error prone | 1 |
| UseEqualsToCompareStrings | Medium (3) | Error prone | 1 |
| SimplifyBooleanReturns | Medium (3) | Design | 1 |
| ForLoopCanBeForeach | Medium (3) | Best practices | 1 |
| UnnecessaryImport | Medium low (4) | Code style | 1 |

## A.1.5

**UncommentedEmptyMethodBody**

This warning has the highest frequency of all violations and also has a medium severity.

**Logo

Description automatically generated**

This warning tells us that there is a method with no statements and no comment explaining why it is empty. Without a comment, it is hard to tell if the method is supposed to be empty or if it has accidentally been left empty. If the method is not supposed to be empty the program will not function as intended and could result in bugs.

To fix the issue you would just need a short comment stating that this method is supposed to be blank and is used to override the methods from the implemented interface.

**CloseResource**

This warning has the second highest frequency and second highest priority.

Text

Description automatically generated

This warning tells us that the java.sql.ResultSet resource needs to be closed after use, as improper closing of resources can result in resource leak. Resource leak is the failure of a program to release a resource after using it, preventing other processes from acquiring said resource, and potentially causing system slowdown or crashing.

The ResultSet resource can be properly closed by moving the declaration and initialisation of “cursor” to inside the *try* block alongside the “conn” and “pstmt” resource declaration and initialisations. Since this is a *try-with-resource* block, any resources declared and initialised inside the *try* block will be automatically closed after execution, without the need to manually close the resources inside a *finally* block.

**LiteralsFirstInComparisons**

Like the previous warning, this warning has the second highest frequency and second highest priority.

Text

Description automatically generated

This warning tells us that literals should be placed first when doing string comparisons to avoid NullPointerExceptions. In this code fragment if “databaseURL” is null we will get a NullPointerException, however, if we swap it with “” the program will be able to read “”, see that the second argument is null and therefore not equal to “”, and return false. If they are not swapped, we would need to add another check to test that the value is not null in order to prevent the NullPointerException.

## A.1.6

**UncommentedEmptyMethodBody**

**CloseResource**

**LiteralsFirstInComparisons**

# CodeQL Analysis

## A.1.7

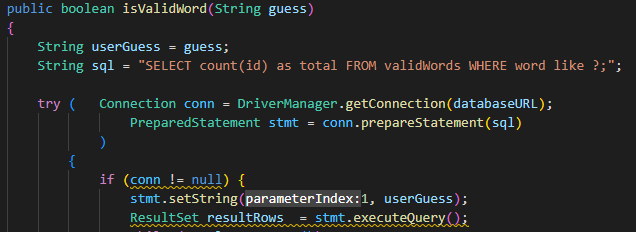
**High severity: Query built by concatenating with a possibly-untrusted string**

**Text

Description automatically generated**

Using string concatenation with direct user input to build SQL queries is a major vulnerability, as it allows users to inject malicious code into the query to gain control of the database. SQL queries should instead be constructed with a wildcard (?) which is given a value using *setString()*. This parameterizes the string input to remove unintended or exploitative effects of special characters.

Here is an example of how to fix the issue:



The concatenated “guess” is replaced by a wildcard (?) which is later replaced with the parameterized user input string using *setString()*.

This vulnerability is a major threat so I would definitely implement a parameterized prepared statement to prevent SQL injection attacks.

**Error severity: No clone method**

**Text

Description automatically generated**

The rectangle class implements the cloneable interface, which by convention, requires that the protected *Object.clone()* method be overridden with a public method when implemented in a class. Since *Cell* extends *Rectangle,* it also implements cloneable and requires that the *clone()* method be overridden. The *clone()* method requires that certain properties must be met for an object to be considered cloned, the result of which may behave unexpectedly when the *clone()* method is not properly overridden. To fix this we must create a *public Cell clone()* method in the *Cell* class that returns *(Cell)super.clone().* I probably wouldn’t do this since the program doesn’t use the *clone()* method, however if I did need to clone anything I would implement the change to avoid any undesired behavior.

**Warning severity: Reference equality test on strings**

****

Using != to compare strings compares the reference value of the objects, not the actual value of the strings, and will therefore not work as intended. In this instance the argument will always evaluate to true even if the value of “databaseURL” is “”. To fix this issue we can use *equals* to compare the contents of the strings, for example *databaseURL.equals(“”)* or *“”.equals(databaseURL).* I would fix this error since it can cause major issues, and I would use the second example solution since I wouldn’t need to add a null check for it to work.

**Note severity: Inefficient empty string test**

Text

Description automatically generated

This warning is similar to the LiteralsFirstInComparisons PMD warning, however, CodeQL suggests that instead of placing the literals first in the comparison it is more efficient to use length to check if a string is empty. If we were to use the suggested solution, we would change the *equals* to *databaseURL.length() == 0* to check that “databaseURL” is empty. However, this does not account for the case where “databaseURL” is null and would therefore require a null test. In this scenario I would not use this solution as it would require writing null checks which is less efficient than the PMD LiteralsFirstInComparisons solution.

# Comparison

## A.1.8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sonarlint** | **Sonarlint severity** | **PMD** | **PMD severity** | **CodeQL** | **CodeQL severity** |
| S1161 - "@Override" should be used on overriding and implementing methods | Major | - | - | Missing override annotation | Note |
| S1186 - Methods should not be empty | Critical | UncommentedEmptyMethodBody | Medium | - | - |
| S2699 - Tests should include assertions | Blocker | UncommentedEmptyMethodBody | Medium | - | - |
| S4973 - Strings and Boxed types should be compared using "equals()" | Major | CompareObjectsWithEquals/ UseEqualsToCompareStrings | Medium | Reference equality test on strings | Warning |
| - | - | LiteralsFirstInComparisons | Medium | Inefficient empty string test | Note |

In total Sonarlint flagged 21 different types of issues, PMD flagged 10, and CodeQL flagged 11.

Sonarlintt tends to have much more severe categorisation for the severity of errors compared to the other two tools, for example it flagged an issue as critical while PMD flagged the same issue as medium.

# User Defined Queries

## A.1.9

## A.1.10

## A.1.11

## A.1.12

## A.1.13