**CA 3 - Lab 1 – Static Analysis Report**

Daniel Gallagher

L00158616

[L00158616@atu.ie](mailto:L00158616@atu.ie)

7th May 2023

# Introduction

Static code analysis, which concentrates on looking at the source or object code, is a technique for analysing code without running it. For code inspection and issue reporting, several paid and open-source tools, including SpotBugs, PMD, and SonarLint, are available. In this report, I use static code analysis tools to analyse and correct two bugs in a project for a billing system. A straightforward Java project for sales and inventory is the billing system.

# Aims/Objectives

* Identify and analyse two bugs in the Billing System project using static code analysis tools.
* Determine if the identified bugs are actual faults or false positives.
* Provide solutions to fix the bugs if they are actual faults.
* Explain the code changes made to address the issues.
* Improve the overall quality and stability of the Billing System project by addressing the identified bugs.

# Method

In this lab, I ran a static code analysis on the Billing System project using the SonarLint extension and Microsoft Visual Studio Code as the integrated development environment (IDE). By analysing the code base and giving real-time feedback on potential bugs, vulnerabilities, and code smells, SonarLint is a potent static code analysis tool that aids developers in finding and fixing issues in their code.

I added the SonarLint extension to Visual Studio Code and imported the Billing System project to start the analysis. To find two significant bugs, I then ran the analysis across the entire project and looked at the reported issues. The following step was to analyse these bugs, determine whether they were true errors or false positives, and then offer fixes for the problems that were found.

**Analysis of Bug 1**

* **Identification of issue – what is the bug the was found.**

I discovered that the DBConnection() method in the DB.java file lacks a try-catch block for handling the ClassNotFoundException that may be thrown when loading the JDBC driver using SonarLint in Visual Studio Code. As shown in figure 1.

A picture containing text, software, multimedia software, screenshot

Description automatically generated

*Fig 1 – Missing try-catch block in ‘DBConnection()’*

* **Solution to this Bug**

To address this issue, I decided to wrap the Class.forName("com.mysql.jdbc.Driver") and DriverManager.getConnection("jdbc:mysql://localhost/caddey", "root", "") calls in a try-catch block to handle both ClassNotFoundException and SQLException exceptions.

* **Code fix – show and explain code changes.**

I added a try block containing the Class.forName("com.mysql.jdbc.Driver") and DriverManager.getConnection("jdbc:mysql://localhost/caddey", "root", "") calls. If the JDBC driver class is not found, a ClassNotFoundException is thrown, and the exception is caught and handled by printing a stack trace and an appropriate error message. Similarly, if there is an issue connecting to the database, an SQLException is thrown, caught, and handled in the same manner. This can be seen in figure 2.

A picture containing text, screenshot, software, multimedia software

Description automatically generated

*Fig 2 – Fixed ‘DBConnection()’ method with try-catch block.*

**Analysis of Bug 2**

* **Identification of issue – what is the bug the was found.**

Using SonarLint in Visual Studio Code, I noticed that in the getMac() method of the Login.java file, the code does not handle exceptions that could be thrown by InetAddress.getLocalHost() and NetworkInterface.getByInetAddress(ip) methods. This can be seen in figure 3.

A screen shot of a computer program

Description automatically generated with medium confidence

*Fig 3 – Lack of error handling in ‘getMac()’ method.*

* **Solution to this Bug**

To resolve this issue, I planned to wrap the InetAddress.getLocalHost() and NetworkInterface.getByInetAddress(ip) method calls in a try-catch block to handle UnknownHostException and SocketException.

* **Code fix – show and explain code changes.**

I added a try block containing the InetAddress.getLocalHost() and NetworkInterface.getByInetAddress(ip) calls. If an exception occurs when trying to get the local host or network interface, an UnknownHostException or SocketException is thrown, respectively. The catch block will handle these exceptions by printing a stack trace and returning an empty string as the MAC address. This can be seen in figure 4.

A picture containing text, screenshot, software, multimedia software

Description automatically generated

*Fig 4 – Updated ‘getMac()’ method with error handling.*

# Conclusion

In this lab, I was able to examine the Billing System project and find two bugs in the code by using a static code analysis tool. I examined the discovered bugs to confirm that they were genuine errors rather than false positives. I then offered fixes for these bugs in order to raise the software's general quality.

First, a try-catch block for handling ClassNotFoundException and SQLException exceptions was missing from the DBConnection() method of the DB.java file. By including the proper try-catch block and properly handling the exceptions, I was able to resolve this problem.

The second bug was discovered in the getMac() method of the Login.java file, where it was discovered that error handling for UnknownHostException and SocketException was not implemented. I solved this problem by including a try-catch block that would handle these exceptions and return an empty string if any of the exceptions materialized.

By resolving these bugs, I was able to increase the Billing System's stability and dependability while also learning valuable lessons about how to use tools for static code analysis to find and fix potential problems in software projects.