LAB 4 – Static Analysis

Student Name: Eryk Gloginski **(L00157413)**

Course: BSc Computing

Module: Secure Programming

Lecturer: Maria Griffin

Submission Date: 16/03/2023

1. Introduction:

Static analysis is an important method that involves the analysing of source code of an application to identify potential vulnerabilities, coding errors or incorrect logic. It is a method of checking code without actually running any of the code. This type of analysis is used to improve software quality assurance processes and to find problems early in the development cycle of the application. In this lab, we will attempt to perform a simple static code analysis with the help of a tool.

2. Aims:

The aims of this lab are to:

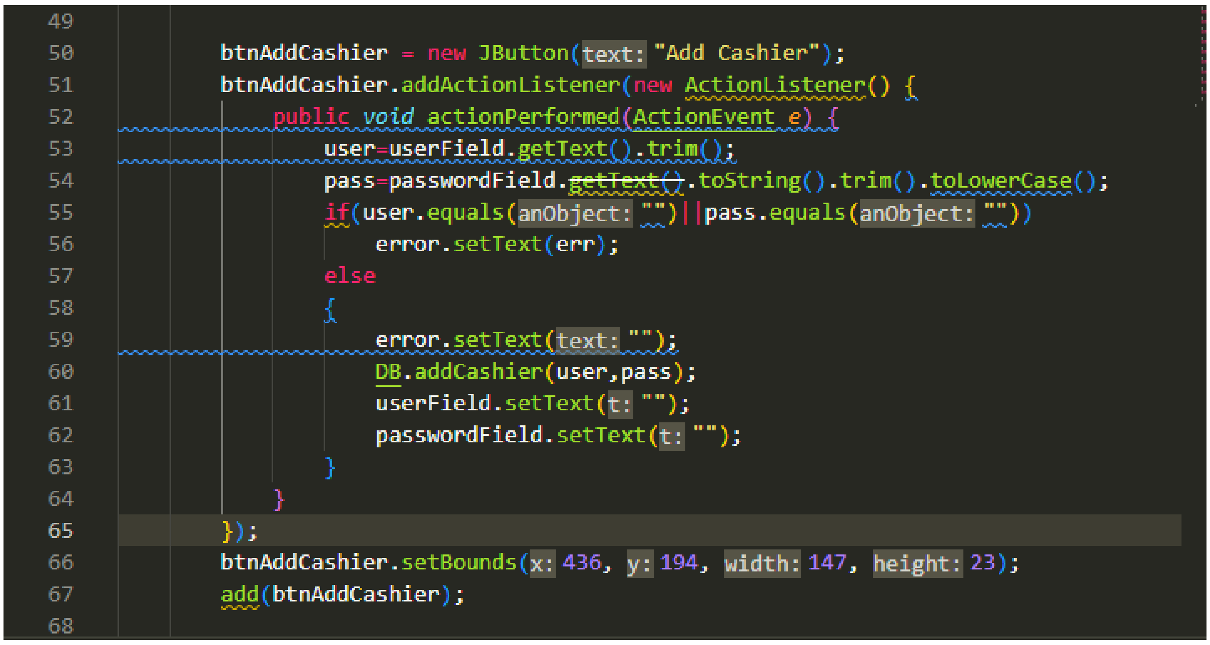
- Set up VS Code (or Eclipse).
- Install SonarLint on VS Code.
- Download and extract the Billing System project code from GitHub.
- Identify any two bugs found by the SonarLint extension and attempt to solve them.

3. Method:

The method used in this lab to perform static analysis is called Linting. Linting is a static code analysis technique that involves the use of a tool called a "linter", to scan the code for potential errors, warnings, and any other issues. Linters usually work by analysing the structure of the code, looking for common errors/issues that can get the code to not work as intended. It is a simple and effective way of performing static analysis on code.

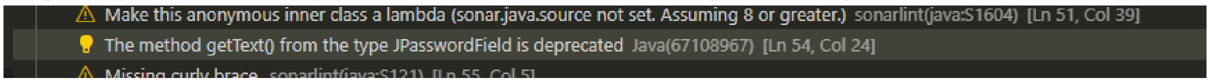
4. Analysis of Bug 1:

In addCashier.java, on line 54, there is a variable called pass that stores the input of the passwordField using getText().



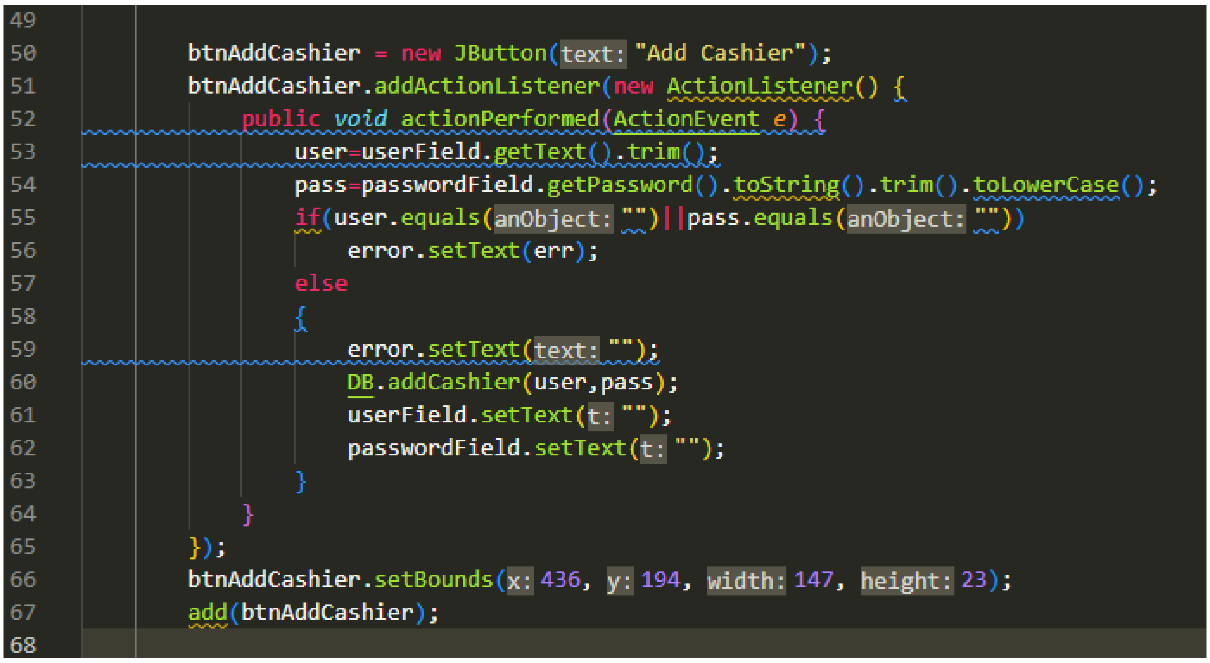
```
49
50          btnAddCashier = new JButton(text: "Add Cashier");
51          btnAddCashier.addActionListener(new ActionListener() {
52              public void actionPerformed(ActionEvent e) {
53                  user=userField.getText().trim();
54                  pass=passwordField.~~getText()~~.toString().trim().toLowerCase();
55                  if(user.equals(anObject: "")||pass.equals(anObject: ""))
56                      error.setText(err);
57                  else
58                  {
59                      error.setText(text: "");
60                      DB.addCashier(user,pass);
61                      userField.setText(t: "");
62                      passwordField.setText(t: "");
63                  }
64              }
65          });
66          btnAddCashier.setBounds(x: 436, y: 194, width: 147, height: 23);
67          add(btnAddCashier);
68
```

By using the SonarLint extension, we can see that the method getText() for the passwordField is deprecated. When a method is deprecated, it means that it should not be used at all, as it is no longer important. It is simply better to remove them as they most likely will not exist in a few years.



```
⚠ Make this anonymous inner class a lambda (sonar.java.source not set. Assuming 8 or greater.) sonarlint(java:S1604) [Ln 51, Col 39]
💡 The method getText() from the type JPasswordField is deprecated Java(67108967) [Ln 54, Col 24]
⚠ Missing curly brace sonarlint(java:S121) [Ln 55, Col 5]
```

Just because a method is deprecated, doesn't mean that it won't work. For example, the getPassword() method is not deprecated and is also a better way of extracting the password. It is simply better to just replace the getText() method with the getPassword() method as the getText() method can suddenly vanish and the code would not work.

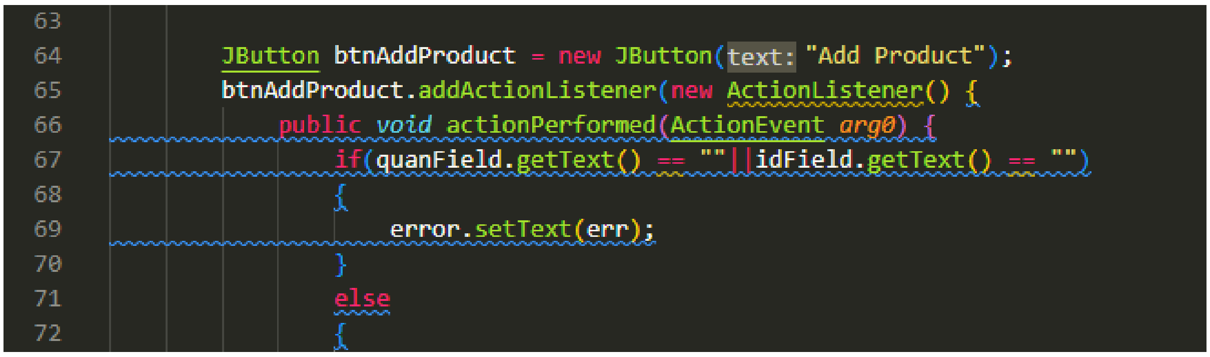


```java

        btnAddCashier = new JButton(text: "Add Cashier");
        btnAddCashier.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                user=userField.getText().trim();
                pass=passwordField.getPassword().toString().trim().toLowerCase();
                if(user.equals(anObject: "")||pass.equals(anObject: ""))
                    error.setText(err);
                else
                {
                    error.setText(text: "");
                    DB.addCashier(user,pass);
                    userField.setText(t: "");
                    passwordField.setText(t: "");
                }
            }
        });
        btnAddCashier.setBounds(x: 436, y: 194, width: 147, height: 23);
        add(btnAddCashier);

```

Finally, I also believe that this "bug" is a false positive as it does not destroy any functionality of the program itself, however, once the getText() method gets removed from the API of the passwordField, it will become an actual fault.

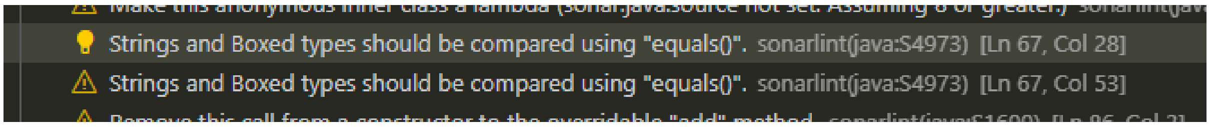
5. Analysis of Bug 2:

In addProduct.java, on line 67, there are two items being compared using ==.



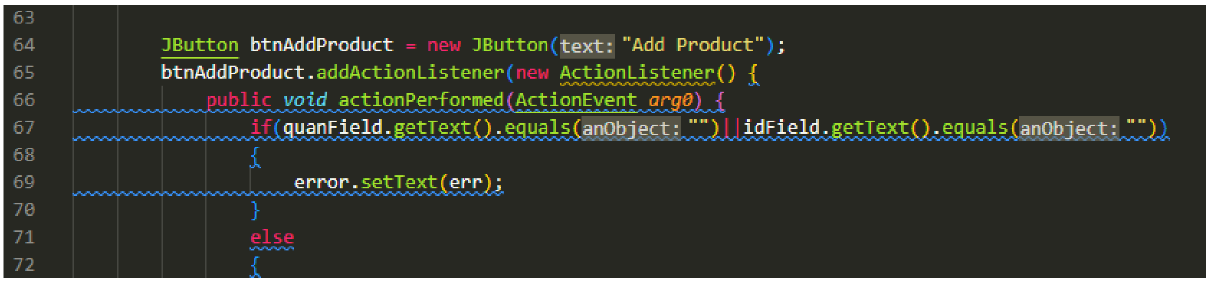
```java
63
64          JButton btnAddProduct = new JButton(text: "Add Product");
65          btnAddProduct.addActionListener(new ActionListener() {
66              public void actionPerformed(ActionEvent arg0) {
67                  if(quanField.getText() == ""||idField.getText() == "")
68                  {
69                      error.setText(err);
70                  }
71                  else
72                  {
```

By using the SonarLight extension, we can see that two primitive items should not be compared using == but rather the equals() method. The == or the != operators compare object references rather than object values. Therefore, they cannot directly compare the values of boxed primitives.



```
⚠ Make this anonymous inner class a lambda (sonar.java.source not set. Assuming 8 or greater.) sonarlint(jav
💡 Strings and Boxed types should be compared using "equals()". sonarlint(java:S4973) [Ln 67, Col 28]
⚠ Strings and Boxed types should be compared using "equals()". sonarlint(java:S4973) [Ln 67, Col 53]
⚠ Remove this call from a constructor to the overridable "add" method. sonarlint(java:S1699) [Ln 86, Col 2]
```

Since boxed primitives cannot be compared using == or !=, we are able to use the equals method to fix this bug.



```

        JButton btnAddProduct = new JButton(text: "Add Product");
        btnAddProduct.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                if(quanField.getText().equals(anObject: "")||idField.getText().equals(anObject: ""))
                {
                    error.setText(err);
                }
                else
                {
```

Finally, I believe that this bug is a fault, not really a false positive, because a false positive could still have the program running. However, when comparing values, you need to make sure that you are comparing the primitive values themself.

6. Conclusion:

In conclusion, static analysis is an important process for software development. The use of static analysis tools such as SonarLint for coding is a great way to improve the quality of your code. The use of such tools is called Linting. It can help developers find errors, help with good coding practices and standards, and find issues early in the development process.

Linting can be customized to meet specific project needs and improve the overall quality and performance of applications. Developers can configure their Linting tool to search for specific errors/issues which saves a lot of time. Overall, static analysis using Linting is a valuable tool for software development and should be used whenever possible.