

**SWT301 Lab2 Report**

**Report – FPT Cinema**

**Lecturer: Đào Thị Thanh**

**Class: SE1762-NJ**

**Group 2**

**Phạm Hồng Phong - HE176151  
Nguyễn Quang Minh - HE176167  
Nguyễn Sỹ Khang - HE176045  
Trần Quang Tùng - HE176091  
Trần Đình Đức - HE176281  
Cao Đức Hiệp - HE171351**

– Hanoi, January 2024 –

# **I. Project Report**

## **1. Status Report**

| **#** | **Work Item** | **Status** | **Notes (Work Item in Details)** |
| --- | --- | --- | --- |
| 1 | Introduction Project | Completed | Phạm Hồng Phong |
| 2 | Introduction to the Testing Tool | Completed | Trần Quang Tùng, Nguyễn Quang Minh |
| 3 | Test Strategy | Completed | Trần Đình Đức, Cao Đức Hiệp |
| 4 | Result Achieved | Completed | All members of group |

## **2. Team Involvements**

| **#** | **Task** | **Member** | **Notes (Task Details, etc.)** |
| --- | --- | --- | --- |
| 1 | T4.3.1 | Phạm Hồng Phong | BUG - S2095: Resources should be closed |
| 2 | T4.3.2 | Phạm Hồng Phong | BUG - S2119: “Random” objects should be reused |
| 3 | T4.3.3 | Phạm Hồng Phong | CODE\_SMELL - S106: Standard outputs should not be used directly to log anything |
| 4 | T4.3.4 | Phạm Hồng Phong | CODE\_SMELL - S1854: Unused local variables should be removed |
| 5 | T4.3.5 | Trần Đình Đức | VULNERABILITY - S6437: Credentials should not be hard-coded |
| 6 | T4.3.6 | Trần Đình Đức | CODE\_SMELL - S1192: String literals should not be duplicated |
| 7 | T4.3.7 | Trần Đình Đức | CODE\_SMELL - S107: Methods should not have too many parameters |
| 8 | T4.3.8 | Trần Đình Đức | CODE\_SMELL - S108: Nested blocks of code should not be left empty |
| 9 | T4.3.9 | Cao Đức Hiệp | CODE\_SMELL - S1104: Class variable fields should not have public accessibility |
| 10 | T4.3.10 | Cao Đức Hiệp | CODE\_SMELL - S1149 : Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used |
| 11 | T4.3.11 | Cao Đức Hiệp | CODE\_SMELL - S4925 : "Class.forName()" should not load JDBC 4.0+ drivers |
| 12 | T4.3.12 | Cao Đức Hiệp | CODE\_SMELL - S116: Field names should comply with a naming convention |
| 13 | T4.3.13 | Nguyễn Sỹ Khang | CODE\_SMELL - S1068: Unused “private”fields should be removed |
| 14 | T4.3.14 | Nguyễn Sỹ Khang | CODE\_SMELL - S1481 Unused local variables should be removed |
| 15 | T4.3.15 | Nguyễn Sỹ Khang | CODE\_SMELL - S3008: static-non-final names should comply with a naming convention |
| 16 | T4.3.16 | Nguyễn Sỹ Khang | CODE\_SMELL - S1170: Public constants and fields initialized at declaration should be "static final" rather than merely "final" |
| 17 | T4.3.17 | Nguyễn Quang Minh | CODE\_SMELL - S1186: Methods should not be empty |
| 18 | T4.3.18 | Nguyễn Quang Minh | CODE\_SMELL - S125: Sections of code should not be commented out |
| 19 | T4.3.19 | Nguyễn Quang Minh | CODE\_SMELL - S1444: "public static" fields should be constant |
| 20 | T4.3.20 | Nguyễn Quang Minh | CODE\_SMELL - S1135: Track uses of "TODO" tags |
| 21 | T4.3.21 | Trần Quang Tùng | CODE\_SMELL - S112: Generic exceptions should never be thrown |
| 22 | T4.3.22 | Trần Quang Tùng | CODE\_SMELL - S1125: Boolean literals should not be redundant |
| 23 | T4.3.23 | Trần Quang Tùng | CODE\_SMELL - S1172: Unused method parameters should be removed |
| 24 | T4.3.24 | Trần Quang Tùng | CODE\_SMELL - S117: Local variable and method parameter names should comply with a naming convention |

## **3. Issues/Suggestions**

| **#** | **Issue** | **Status** | **Notes (Solution, Suggestion, etc.)** |
| --- | --- | --- | --- |
| 1 | Downloaded the wrong version of Sonarlint for netbean13 while using netbeans 17 | Completed | Correct version has been re-downloaded |
| 2 | Don't know how to activate Sonarlint to test code in Netbeans | Completed | Click on Tool –> Plugins –> Installed –> right click on User Installed Plugins –> Activate |

# **II. Introduction Project and Testing Tool**

## **1. Introduction Project**

* **Name**: FPT\_Cinema
* **Purpose**: Users can book movie tickets in cinema, choose type of seats or cancel real-time tickets. System supports create online invoices and make payments
* **User**: Admin, movie ticket booker
* Admin: Manage account login to the website, manage showings and movie rooms in cinema
* Movie Ticket Booker: View movie showtimes, view types of seats in movie rooms, choose to book movie tickets and seat types, view realtime seat status. Tickets create online invoices and make payments. Can view the history booked tickets

## **2. Introduction to the Testing Tool:**

### **2.1 Tool name:** SonarLint

**Description:** SonarLint is a free IDE extension that empowers you to fix coding issues before they exist. Moreover, SonarLint detects and highlights issues that can lead to bugs, vulnerabilities, and code smells as you create your code. It offers clear remediation guidance and educational help, so you can fix issues before the code is committed. SonarLint in VS Code supports analysis of JS/TS, Python, PHP, Java, C, C++, C# and IaC code locally in your IDE.

### **2.2 Purpose of the Testing Tool:**

**Testing objectives:**

* **Code Quality Improvement:** SonarLint aims to improve the overall quality of code by identifying and suggesting fixes for coding standards violations, code smells, and maintainability issues.
* **Early Issue Detection:** The tool helps in the early detection of bugs, security vulnerabilities, and other code issues, preventing them from reaching later stages of the development lifecycle.
* **Consistent Coding Standards:** SonarLint enforces coding standards and best practices, ensuring that the codebase adheres to consistent guidelines.

### **2.3 Level of issue by Solarint**

SonarLint, like many other static code analysis tools, categorizes the severity of issues based on their potential impact on the application's security, performance, or maintainability. The severity levels are usually classified as follows:

* **Blocker**: These issues indicate serious problems that can cause incorrect behavior of the program, are susceptible to external attacks, or can lead to severe crashes in the product. Examples include security vulnerabilities, issues causing application crashes.
* **Critical**: These are issues that usually encompass problems with a significant impact on the application's performance or security, but not as severe as blocker issues. Examples include memory leaks, the use of unsafe APIs.
* **Major**: These issues include problems that can significantly affect the maintainability of the source code and can lead to logical or performance errors. Examples include violations of best practices, non-compliance with coding standards.
* **Minor**: These issues usually involve smaller problems related to code quality that do not seriously affect the functionality, performance, or security of the application. Examples include unclean code writing, violations of coding style standards.
* **Info**: These are informational items about non-error issues that may require attention. These are often suggestions for improving clarity, uniqueness, or performance of the code.

## **3. Test Strategy**

### **3.1. Test approach**

In the white-box testing approach for our project, we delve deeply into the internal structure and design of the code to identify potential issues or defects. The focus will be on improving the quality of the code by ensuring that it meets industry standards and best practices.

We use automated techniques to perform static analysis of the code. Automated techniques will involve using tools like SonarLint to conduct unit tests and scan the code for potential issues related to performance, and maintainability.

### **3.2. Test environment setup**

To effectively implement SonarLint in our test environment, particularly within the Apache NetBeans IDE 17, several critical steps and configurations are necessary. Here's a detailed setup plan:

**Hardware and Software Configurations:**

* **Apache NetBeans IDE:** Verify that all team members are using a compatible version of Apache NetBeans, ideally the latest version to ensure full compatibility with SonarLint.
* **SonarLint Plugin Installation:**

**Plugin Installation:** Install the SonarLint plugin in Apache NetBeans. This can be done through the IDE's plugin manager. Ensure that the installed version of SonarLint is compatible with the version of NetBeans being used.

**Configuration:** Configure SonarLint settings in NetBeans to align with the project's requirements. This includes setting up rules for code quality, security vulnerability detection, and other relevant parameters.

* **Project Configuration:**

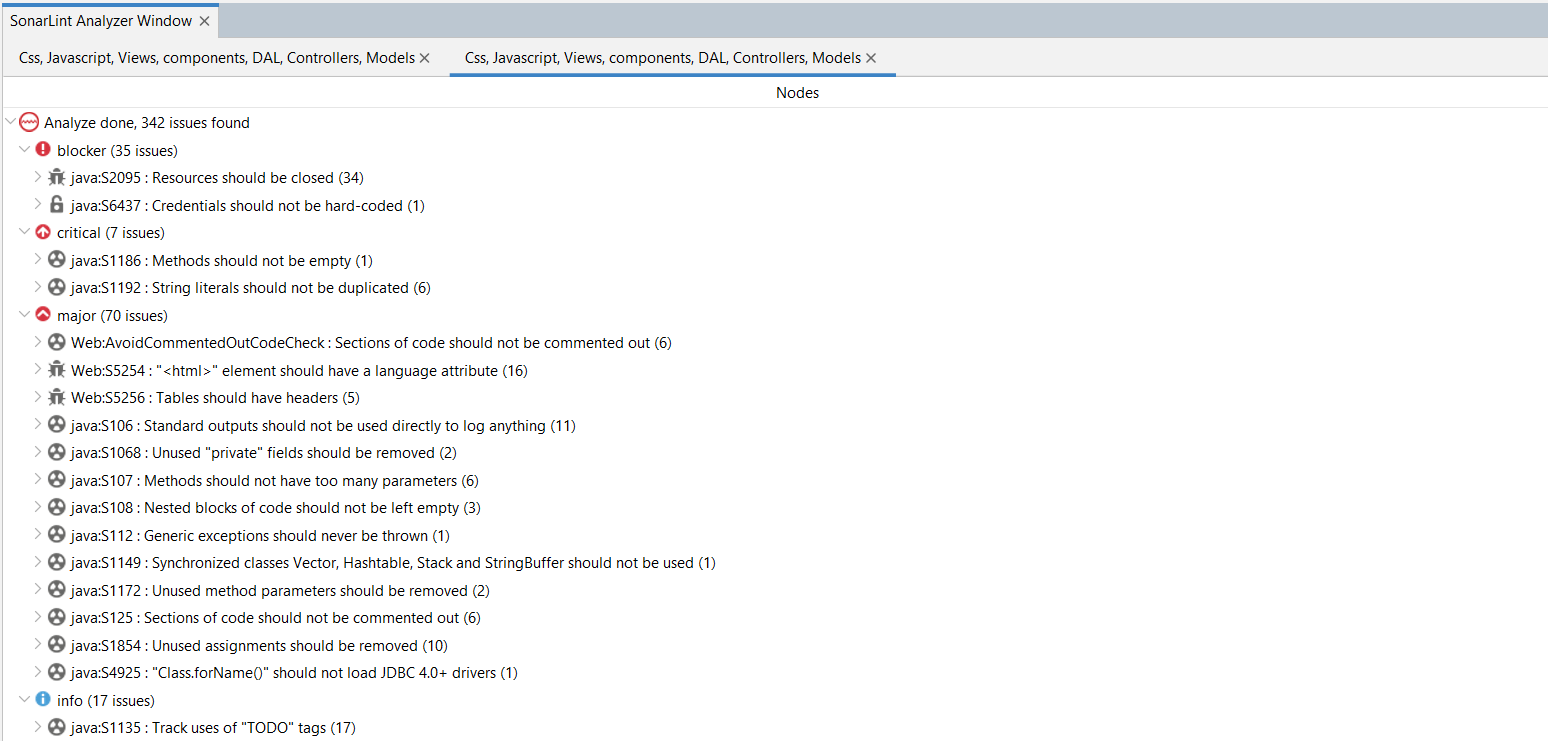
**Dependencies and Libraries:** Confirm that all necessary dependencies and libraries required by the project are correctly configured and accessible in the NetBeans environment.

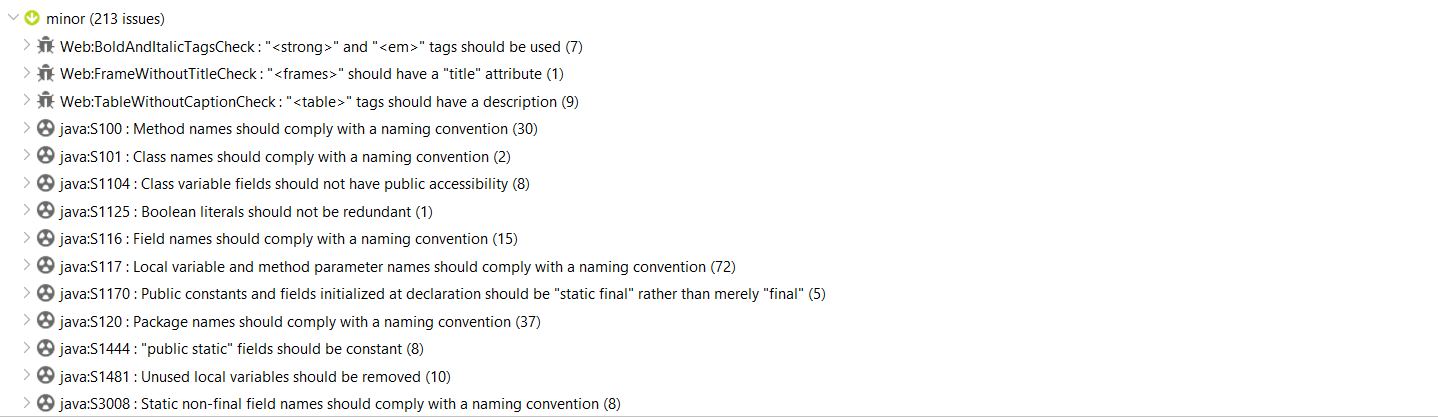
## **4. Result achieved**

### **4.1. Lines of Code (LOC) tested:**

The tool successfully executed test cases on 6668 lines of code in 3 packages (controller, dal, model) and 4 folders (Css, Javascript, Views, components) of the source package of our project.

### **4.2. Issues identified:**

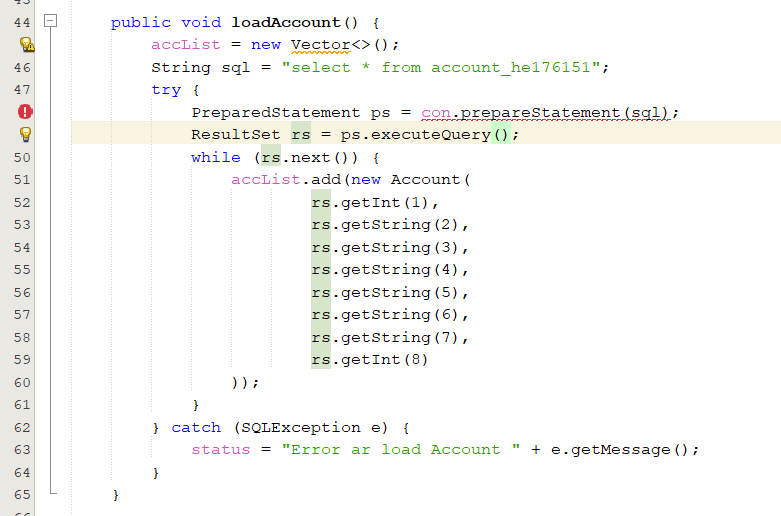


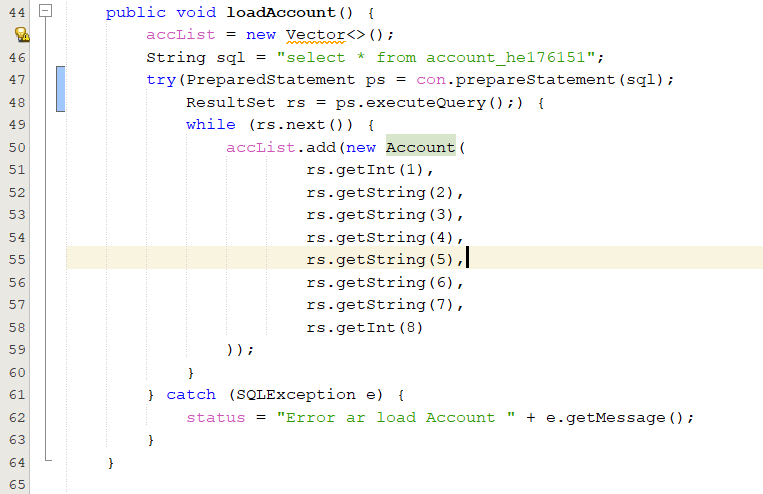


### **4.3. Issue solutions:**

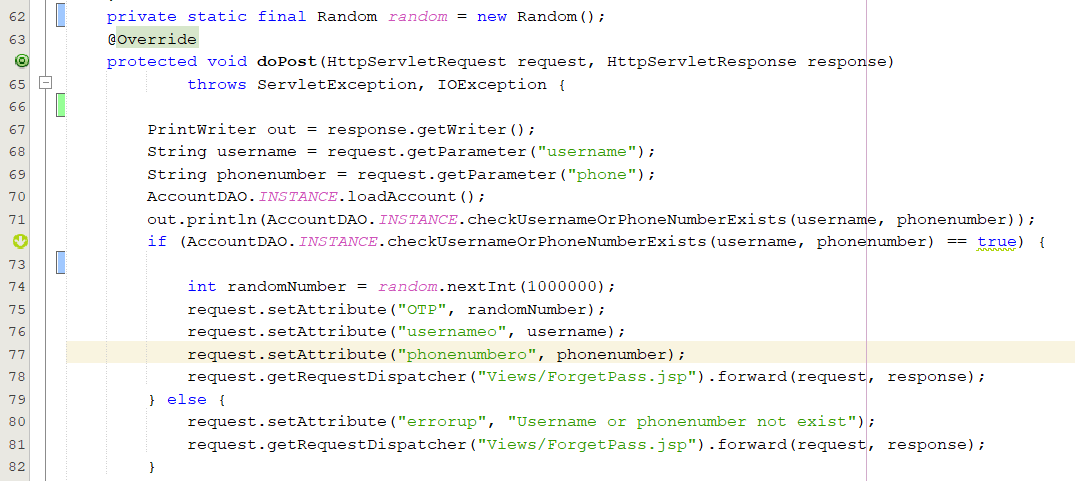
#### **Issue 1:** S2095 - Resources should be closed (Phạm Hồng Phong)

* **Type**: BUG
* **Cause**: By not closing resources such as streams, connections, or files after using them. This is an important issue in resource management.
* **Level**: Blocker
* **Image before fix:**

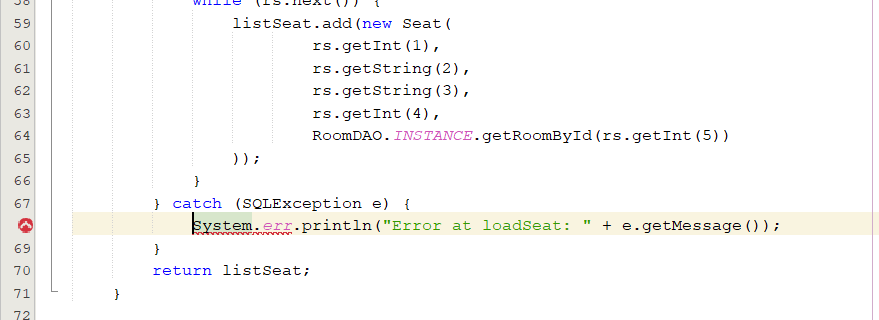
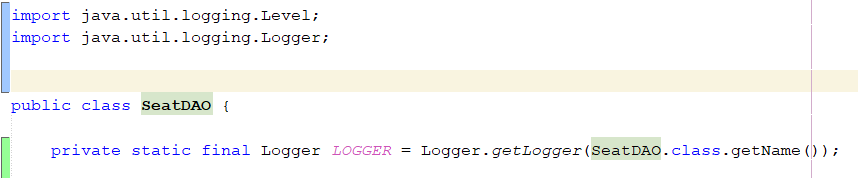


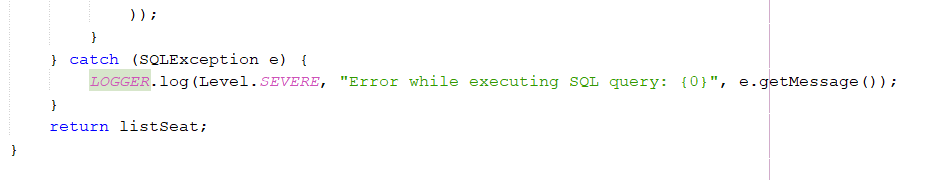
* **Image after fixed:**
* **Lesson learned:** When open the streams, connections or files, we need to close when not using this. Because this will help avoid resource leakage and improve product performance

#### **Issue 2:** S2119 - “Random” objects should be reused (Phạm Hồng Phong)

* **Type**: BUG
* **Cause**: Continuous Creation of New Random Objects, each time to need random numbers, it not only wastes system resources due to the continuous initialization of new objects, but also affects the quality of the generated random number sequence.
* **Level**: Critical
* **Image before fix:**
* **Image after fixed: **
* **Lesson learned:** You should not initialise random every time you want a random number, you should reuse a single Random. This improves performance and reduces the burden on memory and CPU. “Secure Random” can be used to increase security

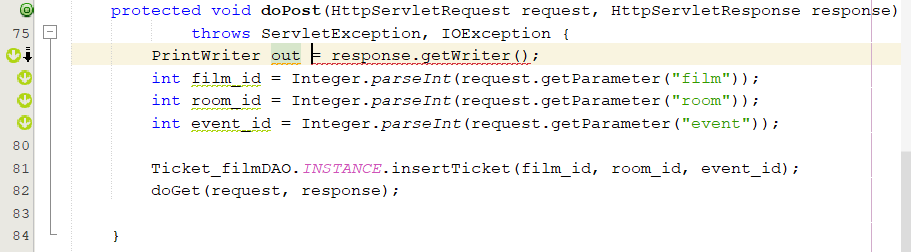
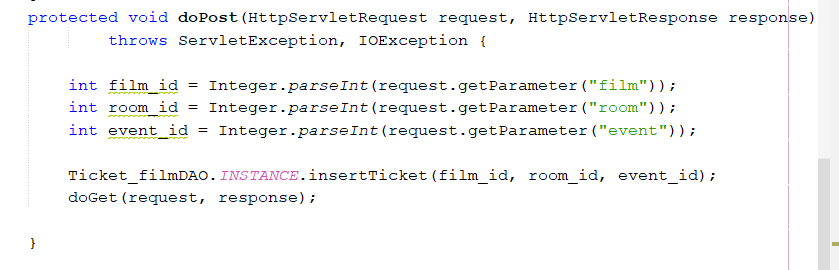
#### **Issue 3:** S106 - Standard outputs should not be used directly to log anything

* **Type**: CODE\_SMELL
* **Cause**: Appears when the source code directly uses standard output streams (System.out or System.err) for logging.
* **Level**: Major
* **Image before fix:**
* **Image after fixed: **

****

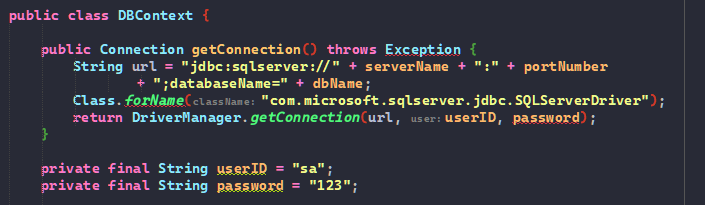
* **Lesson learned:** Should use the java.util.logging library to adjust log granularity, message format, and choose where to send the log. This will help create a solid and manageable code base.

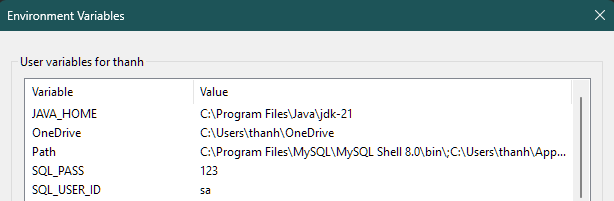
#### **Issue 4:** S1854 - Unused local variables should be removed (Phạm Hồng Phong)

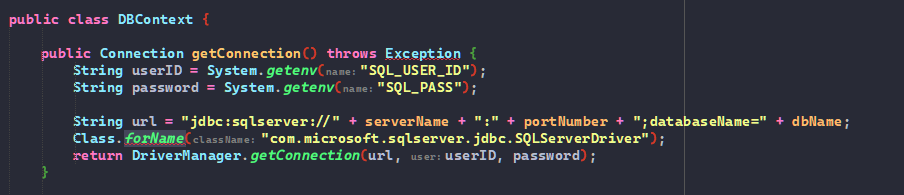
* **Type**: CODE\_SMELL
* **Cause**: initialise the object or assign value while not in use
* **Level**: Minor
* **Image before fix:**
* **Image after fixed:**
* **Lesson learned:** Do not assign a value or initialise an object without using it. This will help prevent other potential errors and make us aware of writing clean code

#### **Issue 5:** S6437 - Credentials should not be hard-coded (Trần Đình Đức)

* **Type**: VULNERABILITY
* **Cause**: The vulnerability identified as S6437 refers to the practice of hard-coding credentials such as user id, password,... within the source code of an application.
* **Level**: Blocker
* **Image before fix:**



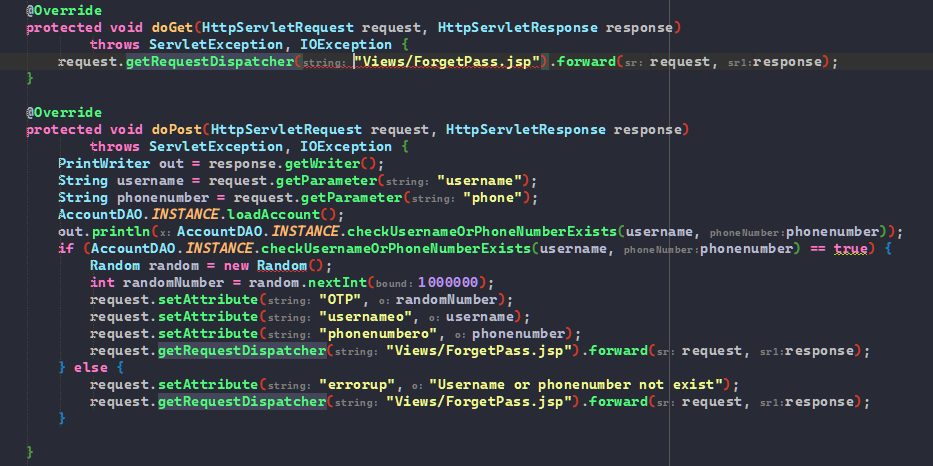
* **Image after fixed:**



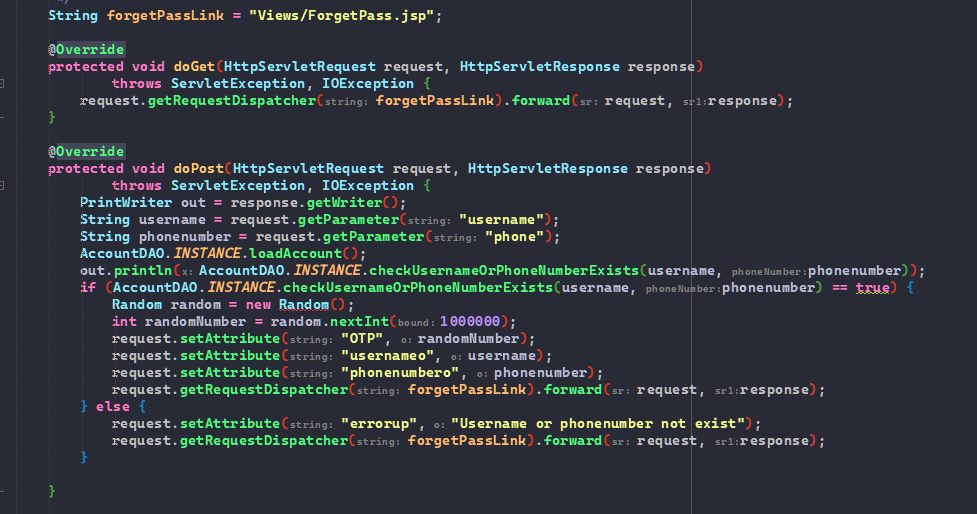
* **Lesson learned:** The lesson that sensitive information such as usernames, user IDs, and passwords should not be hard-coded directly into the source code. Instead, they should be retrieved from external sources through methods or environmental variables, etc.

#### **Issue 6:** S1192 - String literals should not be duplicated (Trần Đình Đức)

* **Type**: CODE\_SMELL
* **Cause**: The "S1192: String literals should not be duplicated" rule in SonarLint is triggered when the same string literal is found duplicated in multiple places within the codebase.The cause of this issue typically stems from a common practice among developers where a string value is directly used in the code instead of being referenced as a constant or an external configuration.
* **Level**: Critical
* **Image before fix:**



* **Image after fixed:**



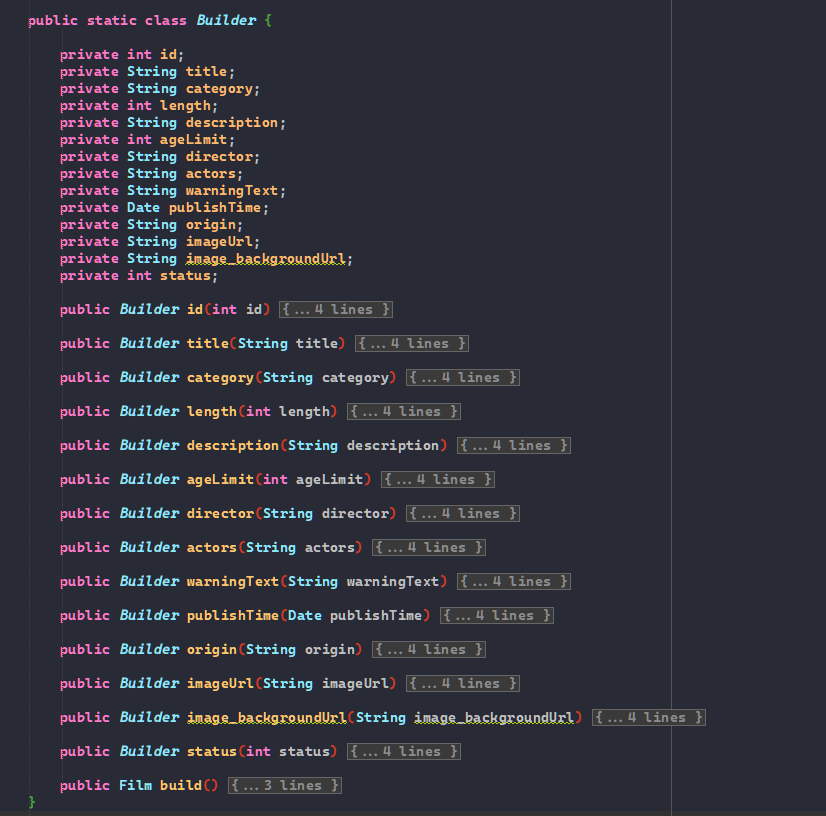
* **Lesson learned:** The lesson learned is that we should avoid repeating the same strings multiple times in the code. Instead, we can use a function that contains the string and call this function whenever needed, rather than rewriting the string. This practice helps to clean up the code by reducing duplication.

#### **Issue 7:** S107 - Methods should not have too many parameters (Trần Đình Đức)

* **Type**: CODE\_SMELL
* **Cause**: SonarLint rule S107, often titled "Methods should not have too many parameters," is triggered when a method or constructor in your code has a large number of parameters.
* **Level**: Major
* **Image before fix:**

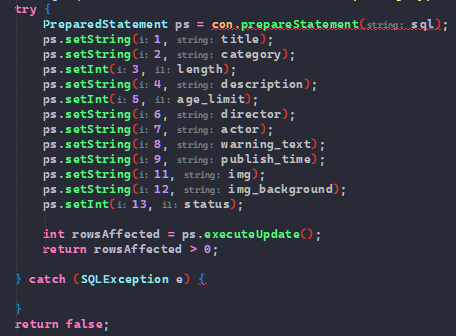


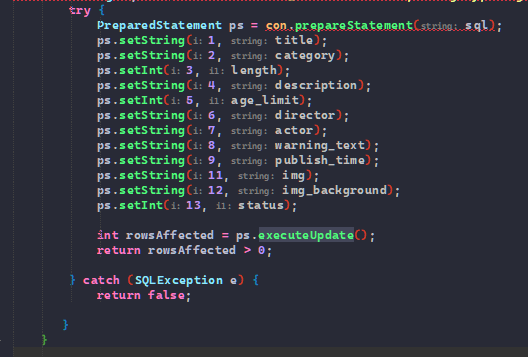
* **Image after fixed:**
* **Lesson learned:** The lesson learned from the principle that methods should not have too many parameters is about maintaining simplicity and clarity in code design. When a method has too many parameters, it can become difficult to understand and maintain. This complexity can lead to errors, as it's harder to track the use and interaction of all parameters.



#### **Issue 8:** S108 - Nested blocks of code should not be left empty (Trần Đình Đức)

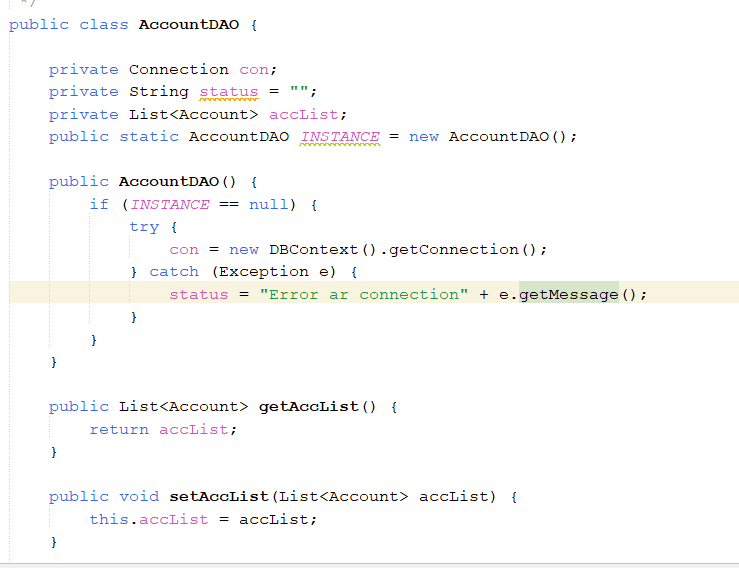
* **Type**: CODE\_SMELL
* **Cause**: The SonarLint rule S108, often titled "Nested blocks of code should not be left empty," is flagged when there are blocks of code, typically within loops, if-else statements, or try-catch blocks, that are left empty without any implementation or even a comment explaining why they are empty.
* **Level**: Major
* **Image before fix:**



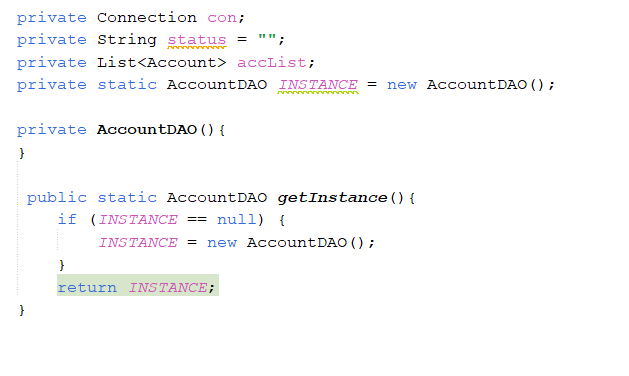
* **Image after fixed:**
* **Lesson learned:** The lesson learned from the guideline that "Nested blocks of code should not be left empty" is important for both code quality and maintainability. Empty nested blocks, such as those in if-else statements, loops, or try-catch blocks, can be misleading, as they imply that a specific action or response is intended, but none is implemented. This can cause confusion for anyone maintaining or reviewing the code, leading to potential misunderstandings or oversights.

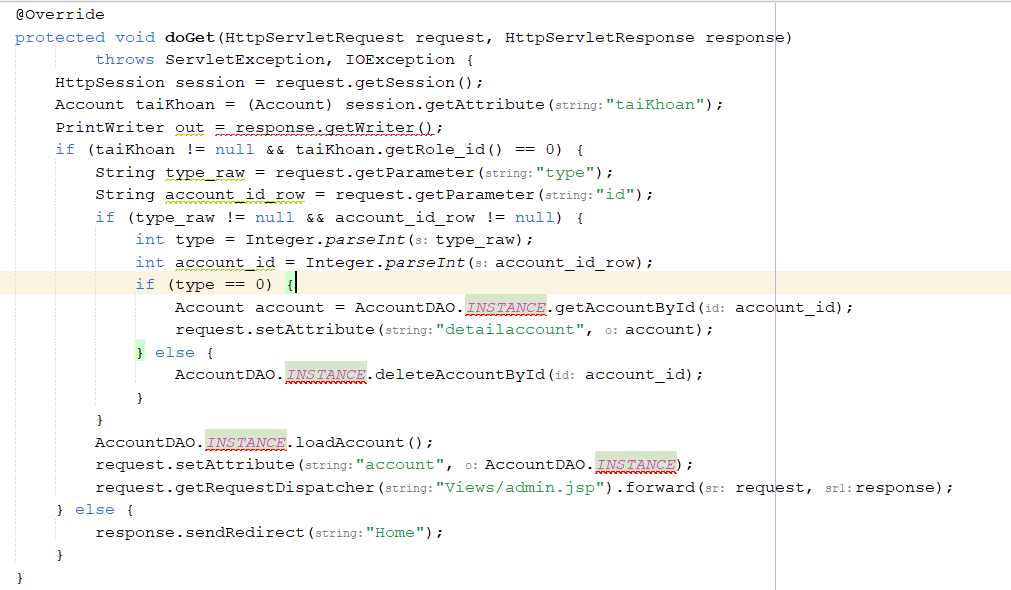
#### **Issue 9:** S1104 - Class variable fields should not have public accessibility (Cao Đức Hiệp)

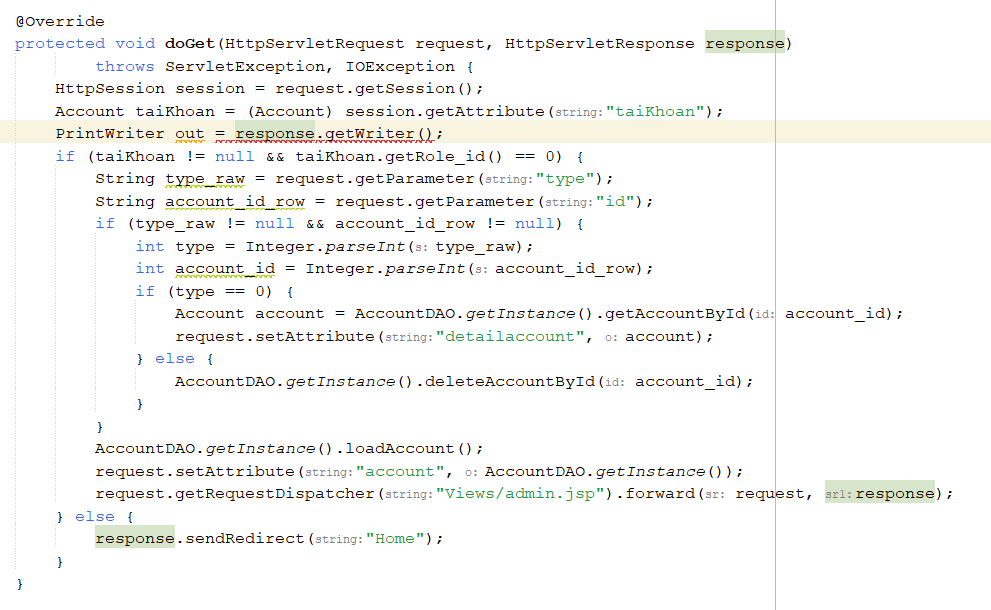
* **Type**: CODE\_SMELL
* **Cause**: INSTANCE variable is public accessibility in class
* **Level**: Minor
* **Image before fix:**

****

* **Image after fixed:**

****

Then, use getInstance() method instead of calling INSTANCE directly.



* **Lesson learned:** When use Singleton pattern, the only object in the class must have private access, the constructor must be private to limit object construction from other classes and a public static method that returns an object of the class, this is the access point so that external classes can get objects from the Singleton class.

#### **Issue 10:** S1149 - Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used (Cao Đức Hiệp)

* **Type**: CODE\_SMELL
* **Cause**: Variable “films”
* **Level**: Major
* **Image before fix:**

****

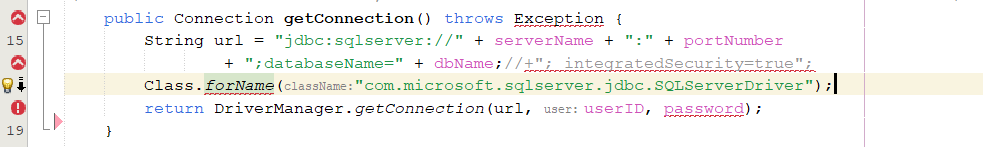
* **Image after fixed:**

****

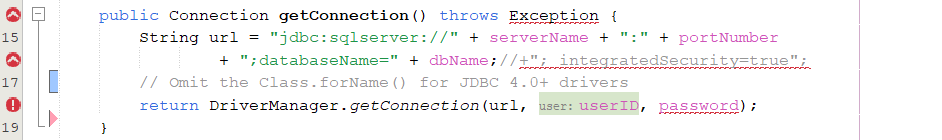
* **Lesson learned:** non-synchronized methods such as ArrayList in single-threaded programs should be chosen instead of Vector because they will help the program run faster.

#### **Issue 11:** S4925 - "Class.forName()" should not load JDBC 4.0+ drivers (Cao Đức Hiệp)

* **Type**: CODE\_SMELL
* **Cause**: “Class.forName()" should not load JDBC 4.0+ drivers
* **Level**: Major
* **Image before fix:**

****

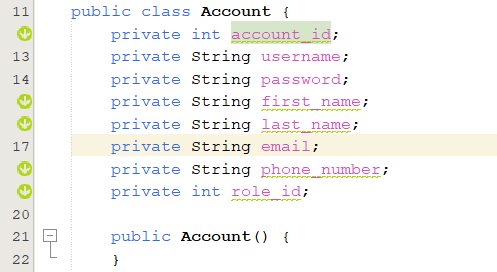
* **Image after fixed:**

****

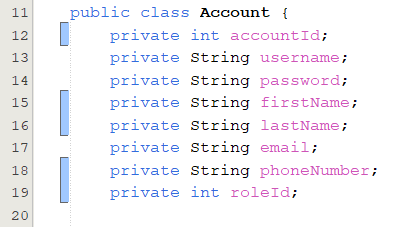
* **Lesson learned:** JDBC 4.0 drivers are discovered automatically. Therefore we no longer need to invoke Class.forName(). Having such a call makes it more cumbersome to switch JDBC drivers, because otherwise one would only have to change the classpath. Also, such code triggers S2658 ("Classes should not be loaded dynamically.”)

#### **Issue 12:** S116 - Field names should comply with a naming convention (Cao Đức Hiệp)

* **Type**: CODE\_SMELL
* **Cause**: Field names should comply with a naming convention
* **Level**: Minor
* **Image before fix:**

****

* **Image before fix:**

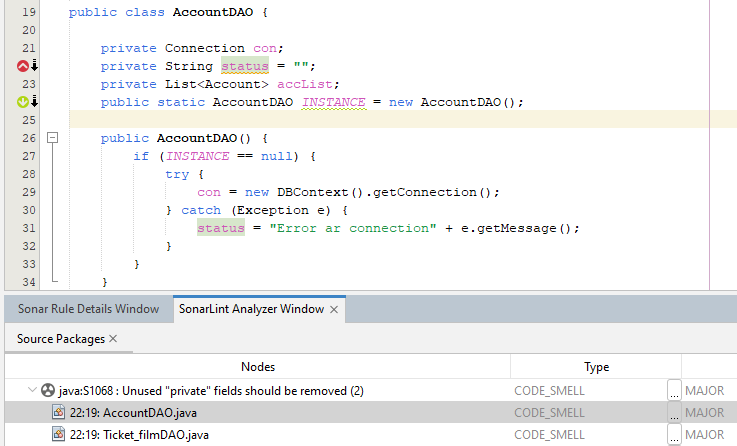
****

* **Lesson learned:** Name convention is a part of code convention Following code convention makes the code easier to read, which helps in teamwork, thereby making the code maintenance and management process faster and easier.

#### 

#### **Issue 13**: S1068 - Unused “private”fields should be removed (Nguyễn Sỹ Khang)

* **Type**: CODE\_SMELL
* **Cause**: This code smell is flagged when private fields in a class are declared but not used in any operations, potentially leading to confusion and unnecessary complexity in the codebase.
* **Level**: Major
* **Image before fix:**



* **Image after fixed:**

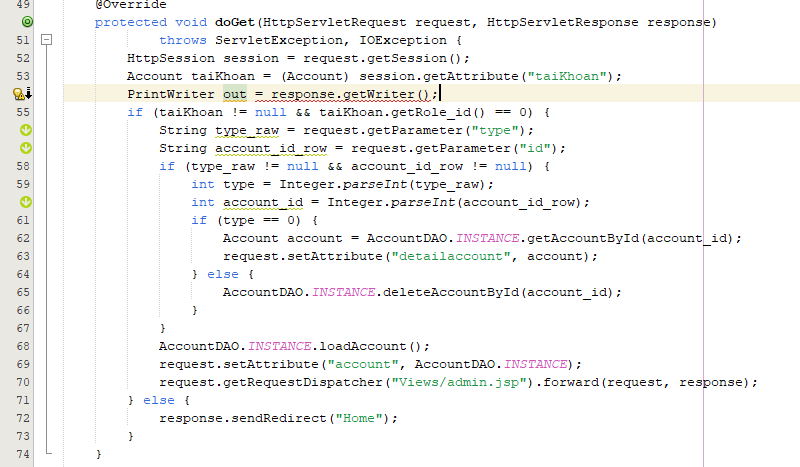
****

* **Lesson learn:**

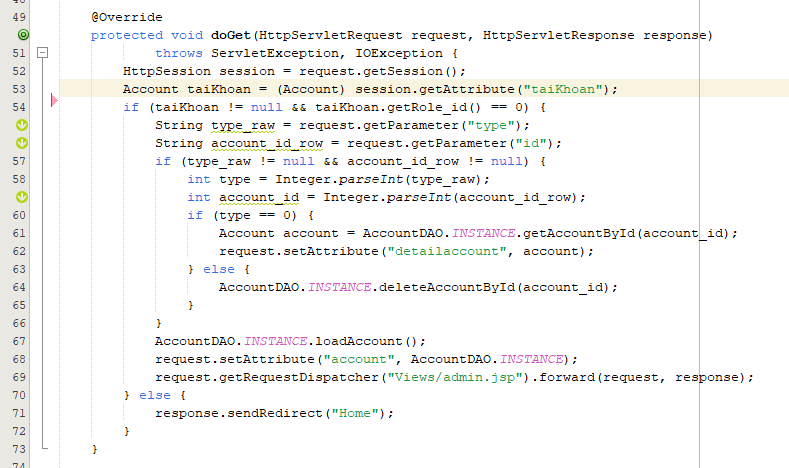
1. The lesson learned from the provided image is that when developing software, it's important to avoid using private fields that are declared but not used (code smell S1068), as they can unnecessarily increase complexity and cause confusion in the codebase.
2. To address this, you should either remove these fields or utilize them meaningfully. Adding "getter" and "setter" methods allows for controlled access and updating of these field values

#### **Issue 14:** S1481 - Unused local variables should be removed (Nguyễn Sỹ Khang)

* **Type**: CODE\_SMELL
* **Cause**: **:**The SonarLint rule S1481, often titled "Unused local variables should be removed," is flagged when This error occurs when a local variable is declared but not used afterwards.
* **Level**: Minor
* **Image before fix:**

****

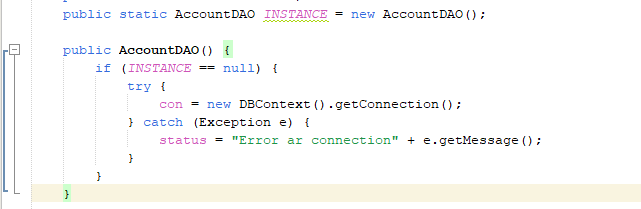
* **Image after fixed:**

****

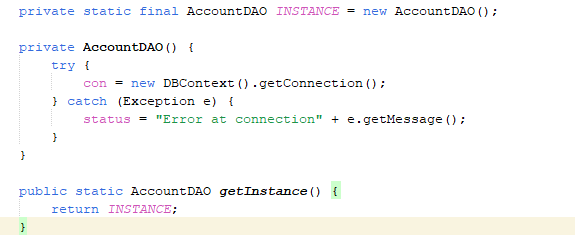
* **Lesson learn:** The lesson learned from the provided image is that we should be careful with declaring unnecessary local variables or variables that are not used. This not only clutters our codebase but can also lead to the unnecessary use of memory. The SonarLint rule S1481 advises that any local variables that are declared but not used afterwards should be removed from the code

#### **Issue 15**: S3008 - Static-non-final names should comply with a naming convention (Nguyễn Sỹ Khang)

* **Type**: CODE\_SMELL
* **Cause**:The SonarLint rule S3008, often titled "static-non-final names should comply with a naming convention" is flagged when The INSTANCE variable in the class is initially declared as static but not final, not following the naming rules for static variables in Java
* **Level**: Minor
* **Image before fix:**



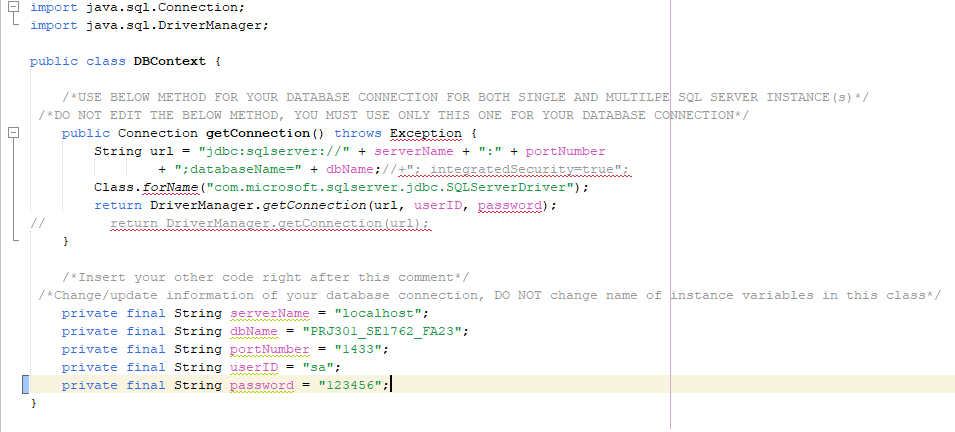
* **Image after fixed:**

****

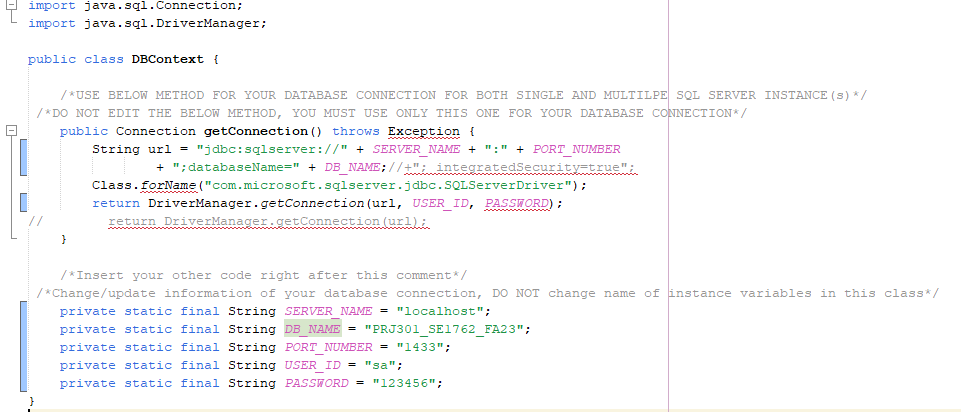
* **Lesson learn:** The lesson learned from the provided image is that all static variables should be declared as final if they do not change after being initialized. This practice enhances the clarity of the code . Additionally, it is important to adhere to naming conventions: static final variables should be named using uppercase letters and underscores to differentiate them clearly.

#### **Issue 16**: S1170 - Public constants and fields initialised at declaration should be "static final" rather than merely "final" (Nguyễn Sỹ Khang)

* **Type**: CODE\_SMELL
* **Cause:** The SonarLint rule S1170, often titled "Public constants and fields initialised at declaration should be "static final" rather than merely "final," is flagged when the error occurs, where a public constant is not marked as static final but only as final, resulting in each instance of the class creating a copy of the constant, wasting memory.
* **Level**: Minor
* **Image before fix**

****

* **Image after fixed:**

****

* **Lesson learned:** The lesson learned from the provided image is that when declaring public constants, we must ensure that they are declared not only as final but also as static. The SonarLint rule S1170 emphasises that public variables initialised at declaration should be marked as static final. This prevents the creation of unnecessary copies of the constant for each instance of the class, saving memory and optimising the application's performance.

#### **Issue 17:** S1186 - Methods should not be empty (Nguyễn Quang Minh)

* **Type**: CODE\_SMELL
* **Cause**: The rule S1186 indicates that a method has been defined but doesn't contain any implementation. Empty methods can lead to confusion, maintenance issues, and can be considered as potential dead code.
* **Level**: Critical
* **Image before fix:**
* ****

* **Image after fixed:**



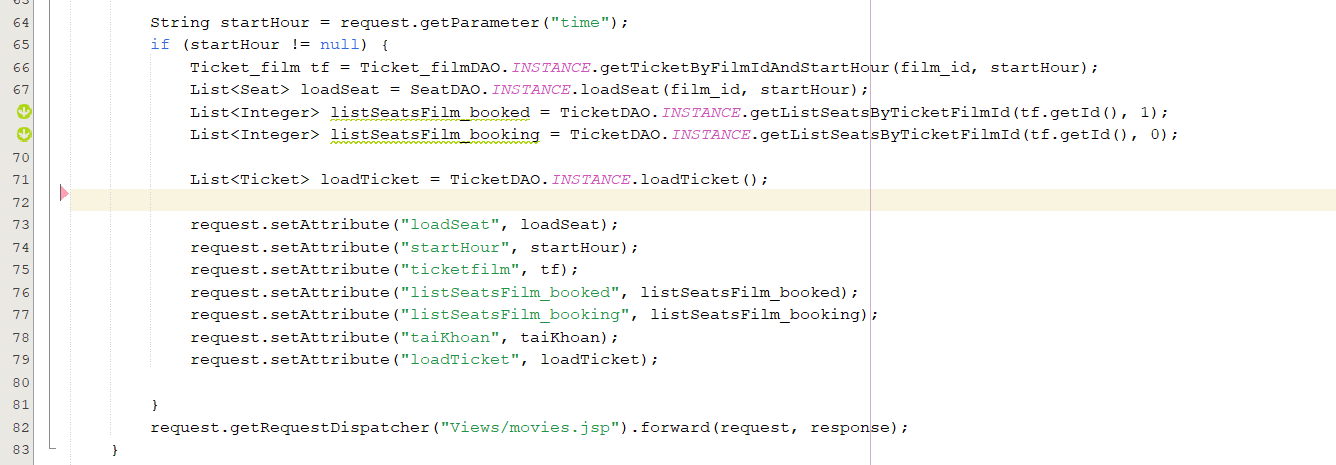
* **Lesson learned:** Through testing and fixing errors S1186 helps me learn that an empty method is often considered as "dead code," which means it doesn't contribute to the functionality of the application. Removing dead code improves code readability and maintainability.

#### **Issue 18:** S125 - Sections of code should not be commented out (Nguyễn Quang Minh)

* **Type**: CODE\_SMELL
* **Cause**: The SonarQube rule S125, often referred to as "Sections of code should not be commented out," indicates that commented-out code sections should be removed rather than being left in the codebase. It's a good practice to eliminate such commented-out code to keep the codebase clean and maintainable.
* **Level**: Major
* **Image before fix:**

****

* **Image after fixed:**

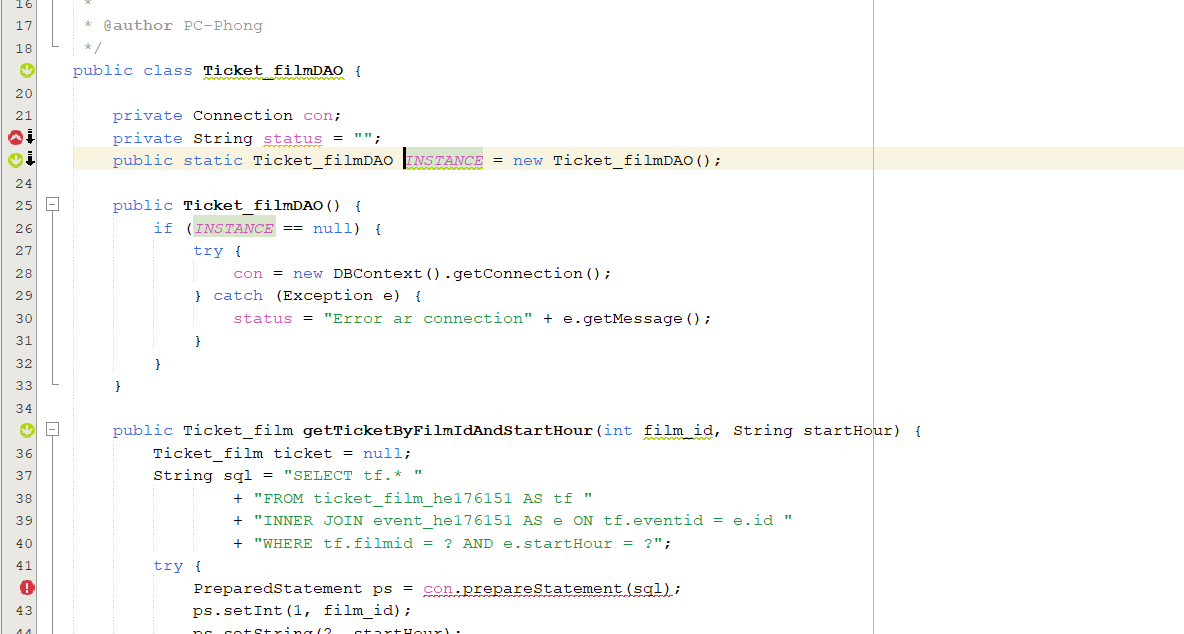


* **Lesson learned:** Through testing and fixing errors S125 helps me learn to prioritize a clean, readable, and maintainable codebase by removing commented-out sections.

#### 

#### **Issue 19:** S1444 - "public static" fields should be constant (Nguyễn Quang Minh)

* **Type**: CODE\_SMELL
* **Cause**: The SonarLint issue S1444, which suggests making the field status final, arises from the fact that status is assigned a value within the constructor and never modified afterward. Making it final ensures that its value is set only once and cannot be changed later.
* **Level**: Minor
* **Image before fix:**



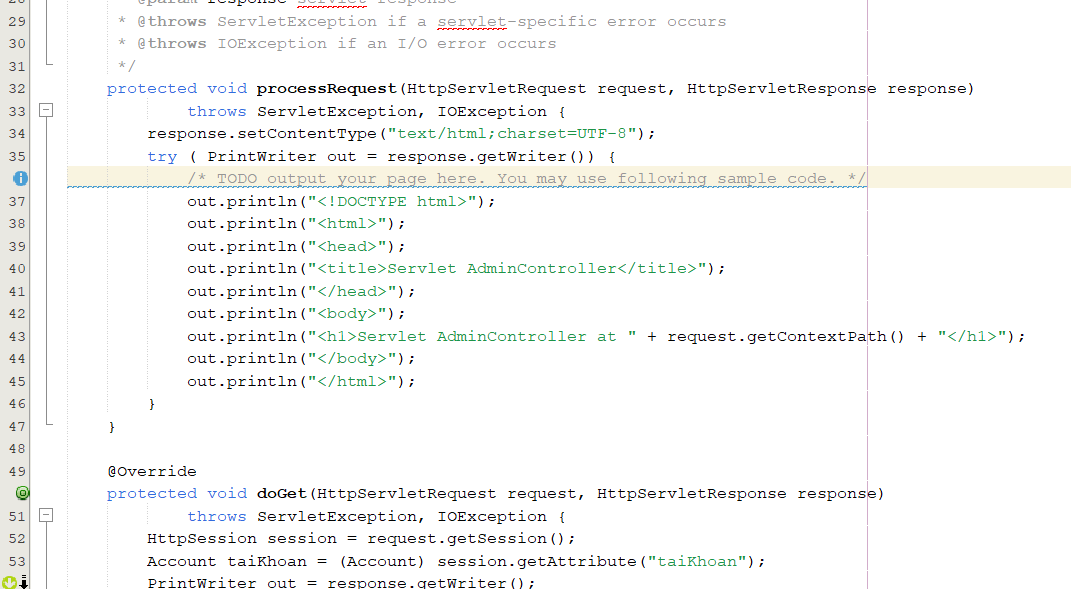
* **Image after fixed:**

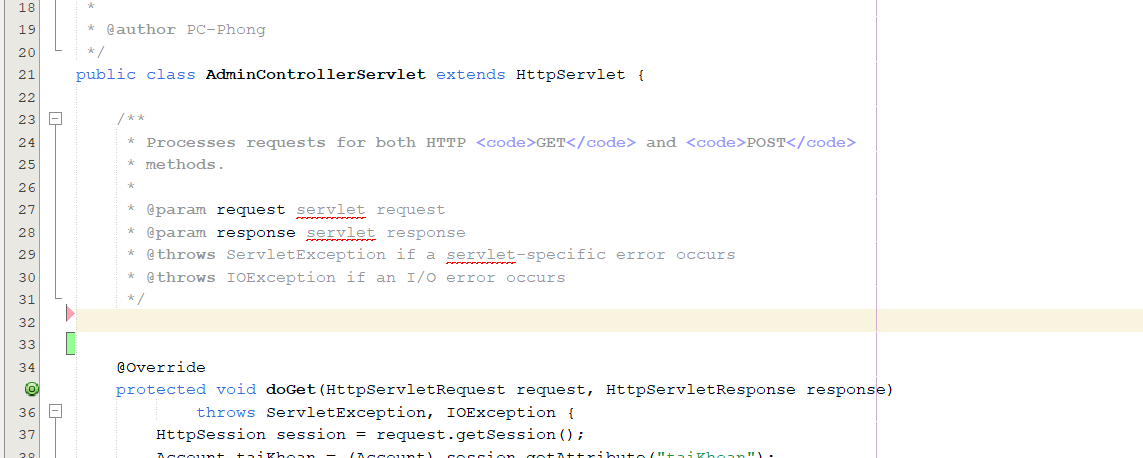


* **Lesson learned:** Through testing and fixing errors S1444 helps me learn that doing so promotes immutability, prevents unintended changes, enhances code clarity, and contributes to overall code quality and maintainability.

#### **Issue 20:** S1135 - Track uses of "TODO" tags (Nguyễn Quang Minh)

* **Type**: CODE\_SMELL
* **Cause**: The TODO comment indicates a placeholder for future work or a reminder to implement something. SonarLint raises an issue to remind developers to revisit and address these comments.
* **Level**: Info
* **Image before fix:**

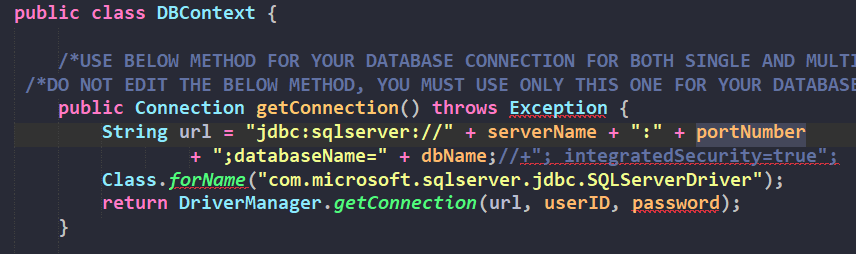


* **Image after fixed:**

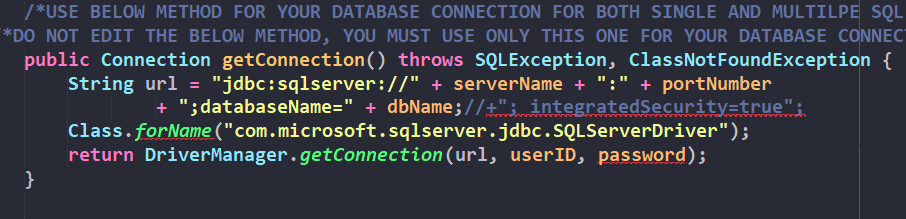
* **Lesson learned:** Through testing and fixing errors S1135 helps me learn that to maintain a clean and readable codebase we need to eliminate methods that will not be needed in the future but have "TODO" comment.

#### **Issue 21:** S112 - Generic exceptions should never be thrown (Trần Quang Tùng)

* **Type**: CODE\_SMELL
* **Cause**: Failure to clearly identify the common throw error like Exception or RuntimeException makes it difficult to accurately identify the error and maintain it.
* **Level**: Major
* **Image before fix:**



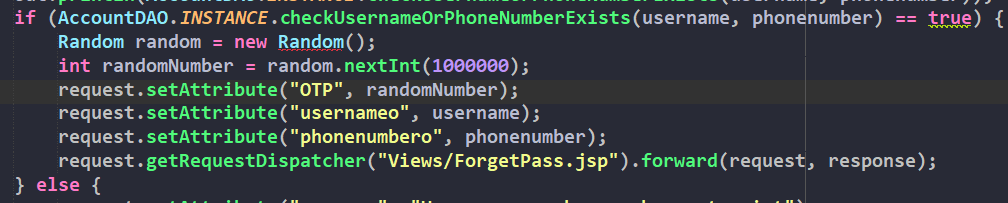
* **Image after fixed:**

****

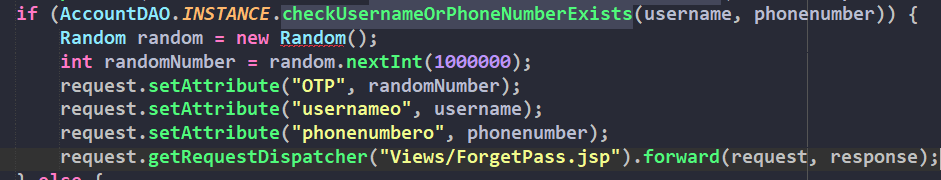
* **Lesson learned:** Through testing and fixing errors S112 helps me learn that I should not throw exception errors in a general way, but need to throw errors in a detailed and specific way to easily clearly identify errors, handle errors and effectively.

#### **Issue 22:** S1125 - Boolean literals should not be redundant (Trần Quang Tùng)

* **Type**: CODE\_SMELL
* **Cause**: Source code can become harder to read when it contains unnecessary comparisons to true or false.
* **Level**: Minor
* **Image before fix:**

****

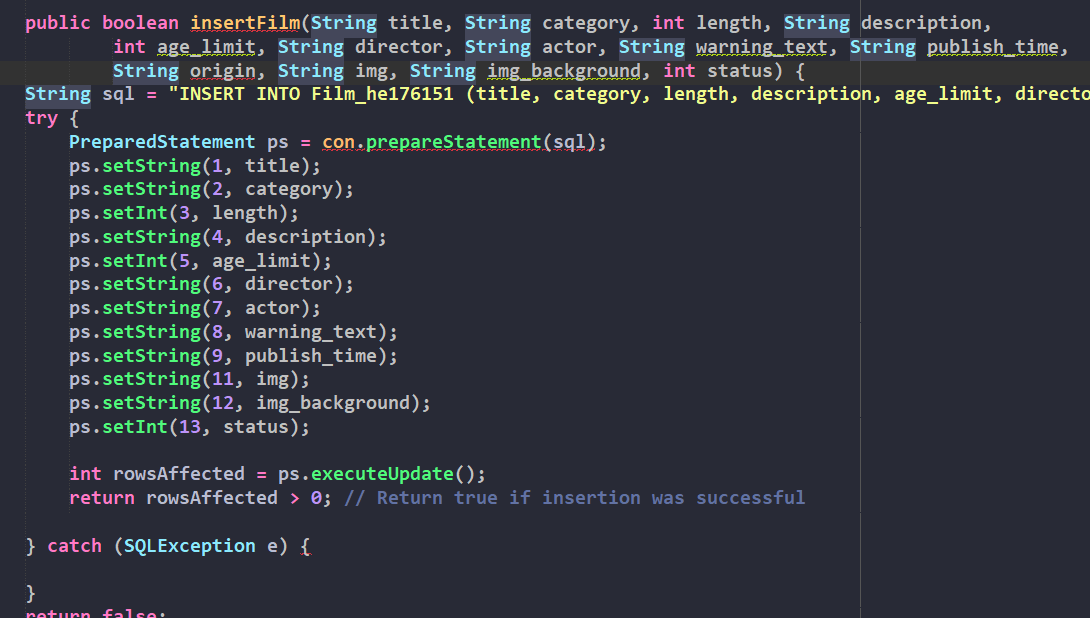
* **Image after fixed:**

****

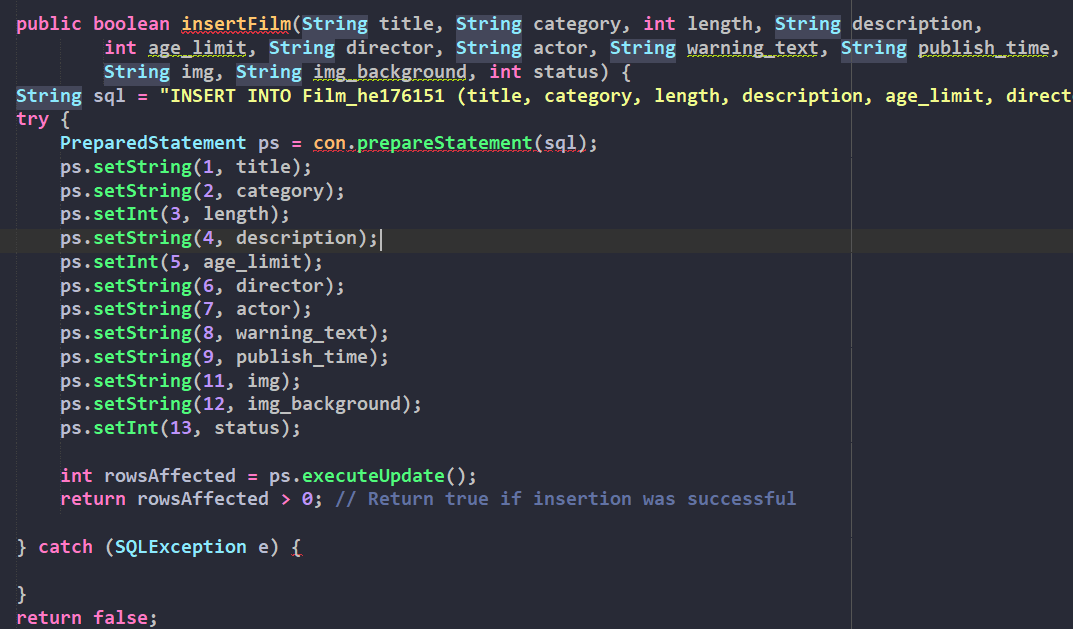
* **Lesson learned**: Through testing and fixing errors S112 helps me learn that I should eliminate unnecessary boolean values, making the source code easier to read and reducing the possibility of misunderstandings.

#### **Issue 23:** S1172 - Unused method parameters should be removed (Trần Quang Tùng)

* **Type**: CODE\_SMELL
* **Cause**: Unused method parameters can clutter the codebase, decrease readability, and potentially lead to confusion.
* **Level**: Major
* **Image before fix:**



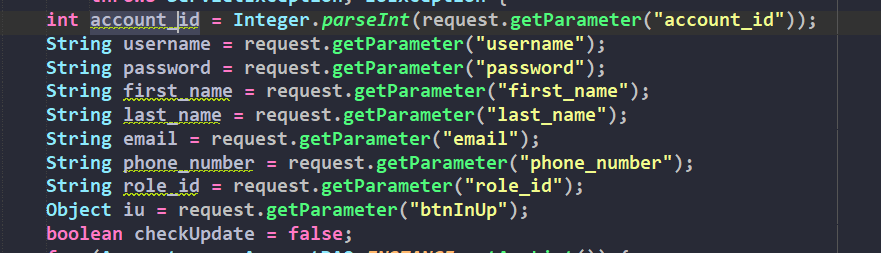
* **Image after fixed:**

****

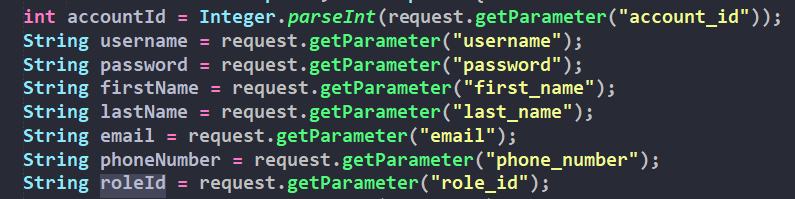
* **Lesson learned**: Through testing and fixing errors S1172 helps me learn that I need to remove unused parameters to help increase the clarity of the source code. This makes my source code easier to read and reduces the potential for confusion for code readers.

#### **Issue 24:** S117 - Local variable and method parameter names should comply with a naming convention (Trần Quang Tùng)

* **Type**: CODE\_SMELL
* **Cause**: Non-compliance with the specified naming convention.
* **Level**: Minor
* **Image before fix:**



* **Image after fixed:**

****

* **Lesson learned**: Through finding and fixing errors S117 helps me learn how to name according to standard rules because when I need to read or understand a piece of code, I will easily determine the meaning of variables or parameters based on their names. Moreover, it helps me easy to read, understand, and maintain