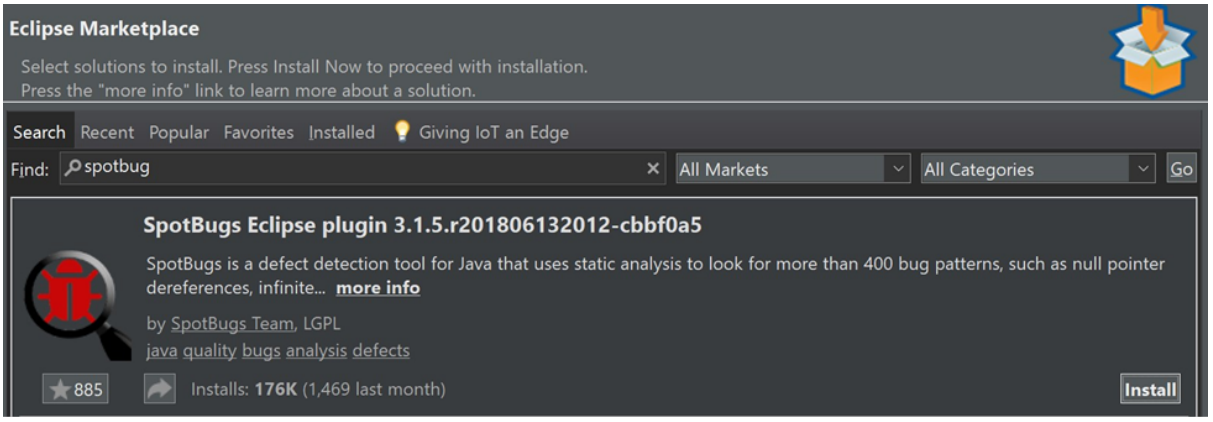
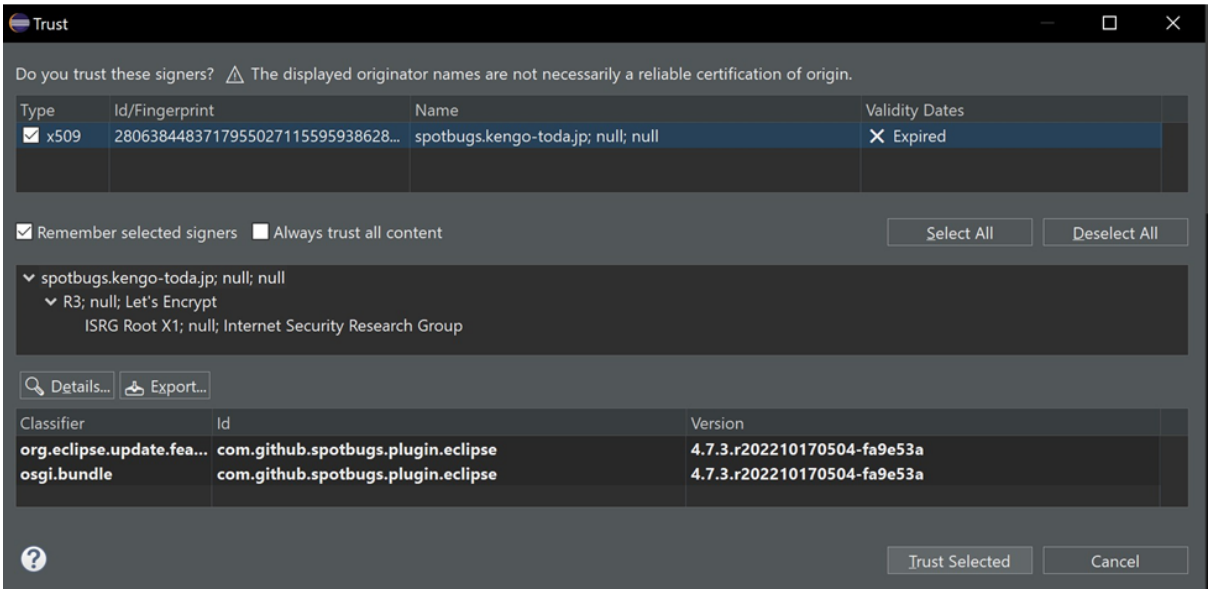
Car Rental System Analysis

Static Analysis

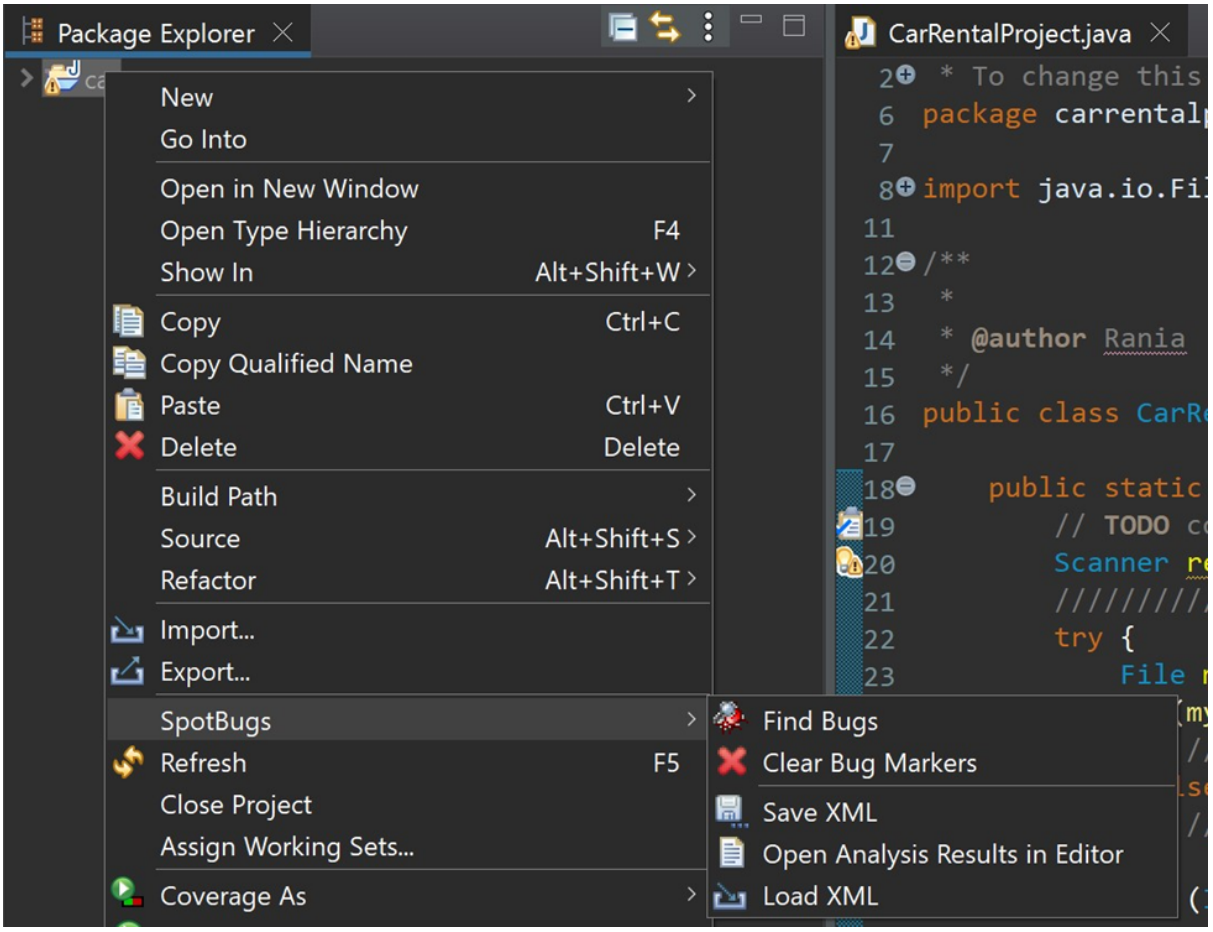
To perform static analysis on the car rental system, I will utilize the SpotBugs tool. SpotBugs is a popular static code analysis tool that helps identify potential bugs, security vulnerabilities, and coding errors in the source code.

The first step is to install SpotBugs from the Eclipse Marketplace or any other suitable method.

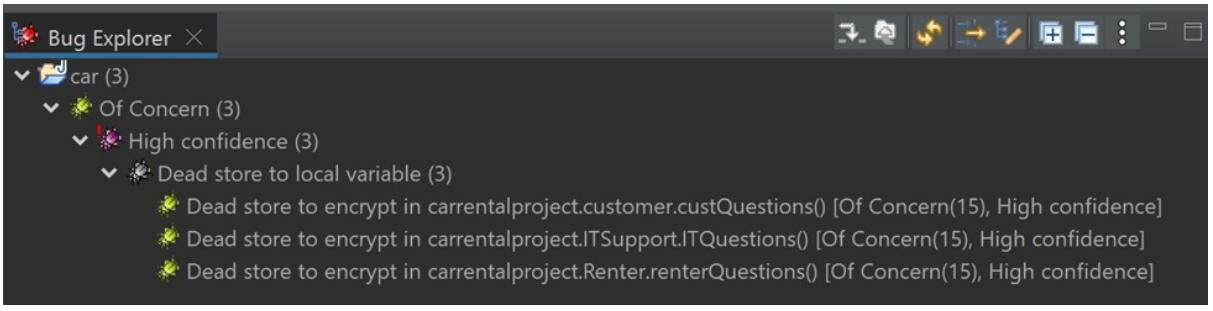




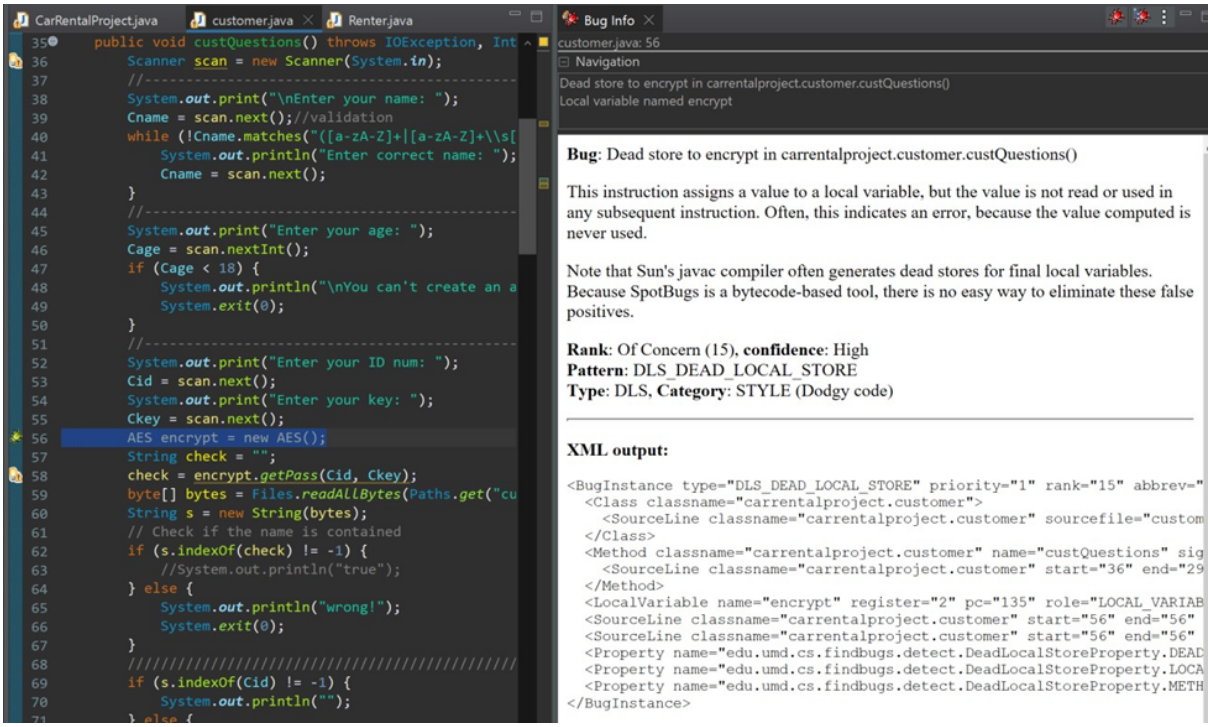
Once installed, I will configure SpotBugs to analyze the program code of the car rental system.



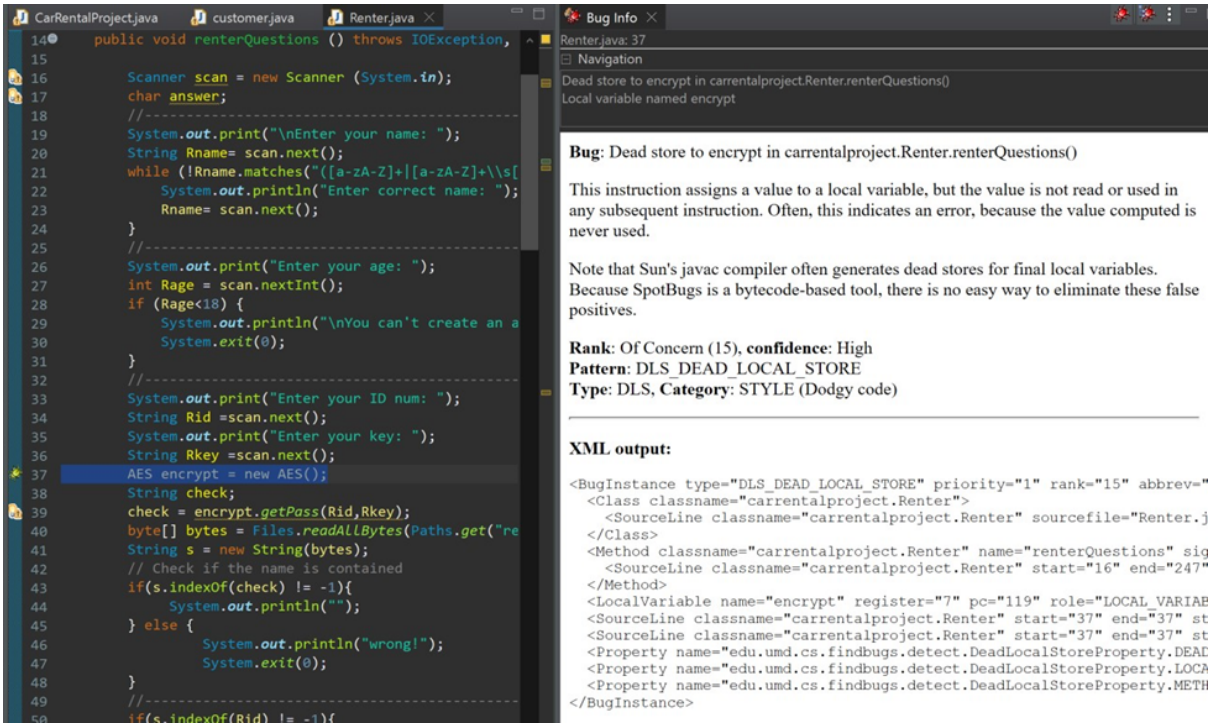
The bugs of concern found using SpotBugs tool



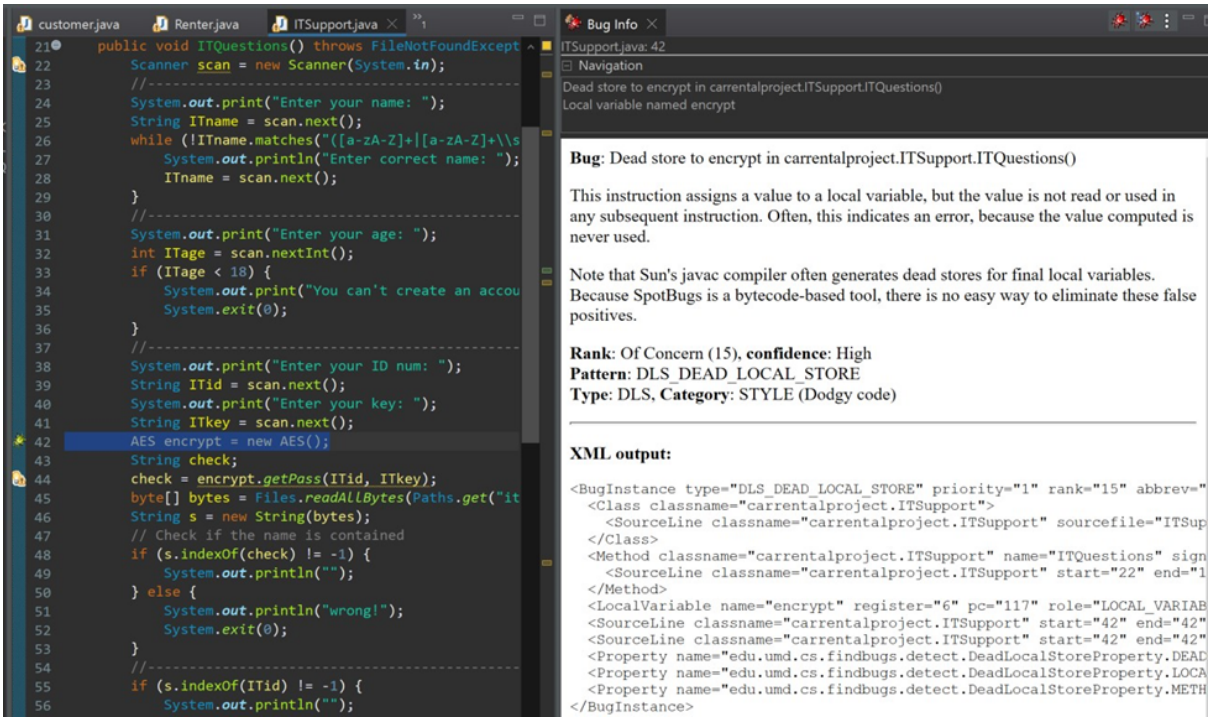
Case 1: in customers class



Case 2: in renters class



Case 3: in ITsupport class



Findings

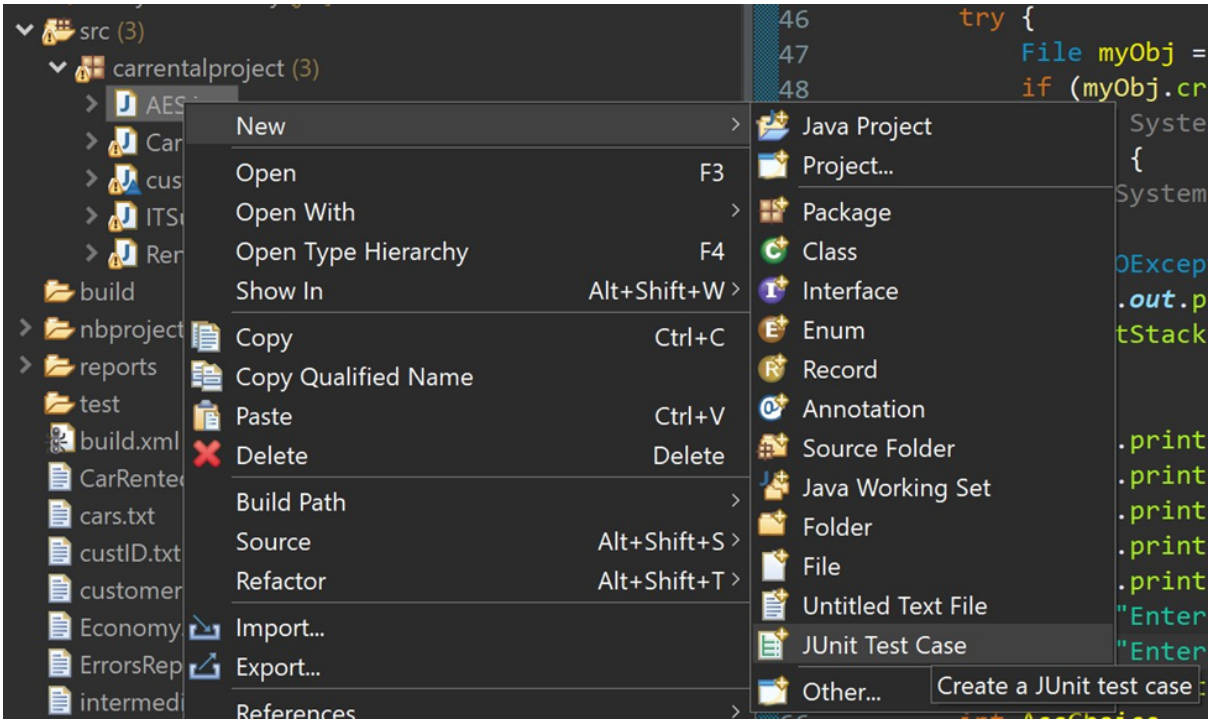
The following findings were discovered through the analysis:

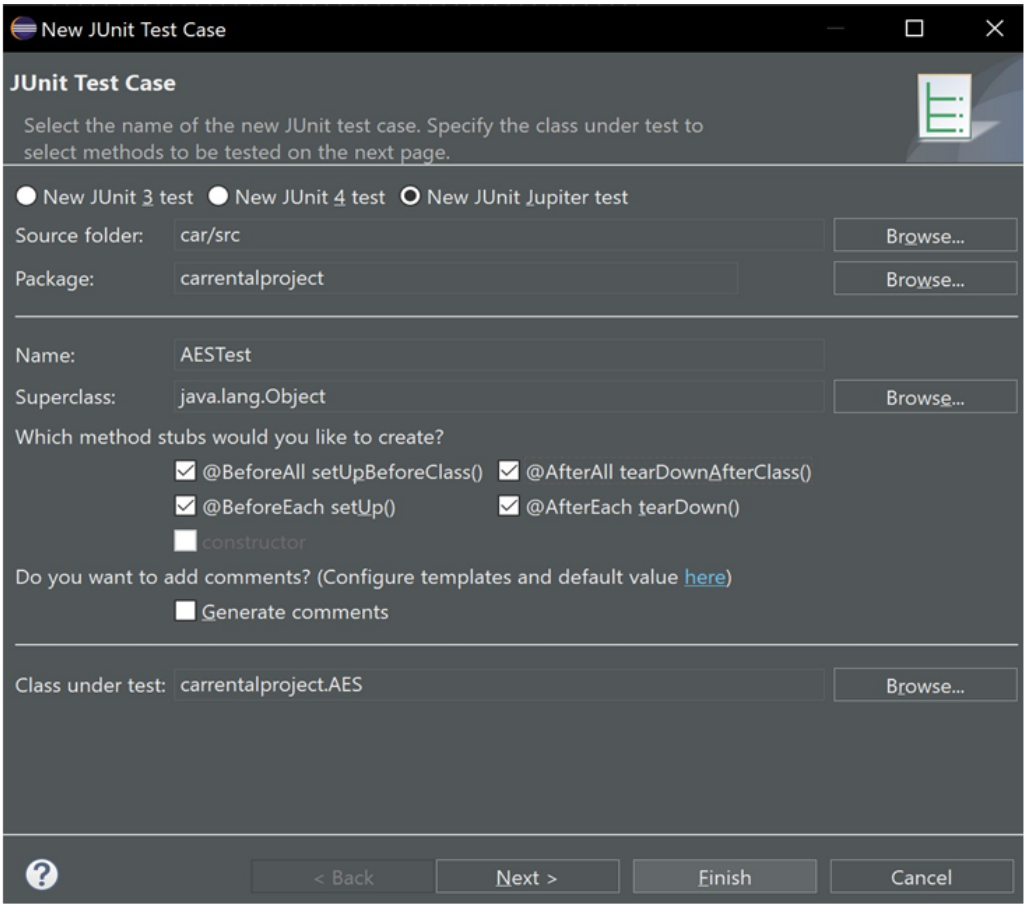
1. In the 3 classes: SpotBugs identified a bug related to a dead store to encrypt. This bug indicates that the encryption operation performed in the code is redundant and does not contribute to the system's functionality. It is recommended to either remove the encryption operation if unnecessary or modify the code to utilize the encrypted value appropriately.

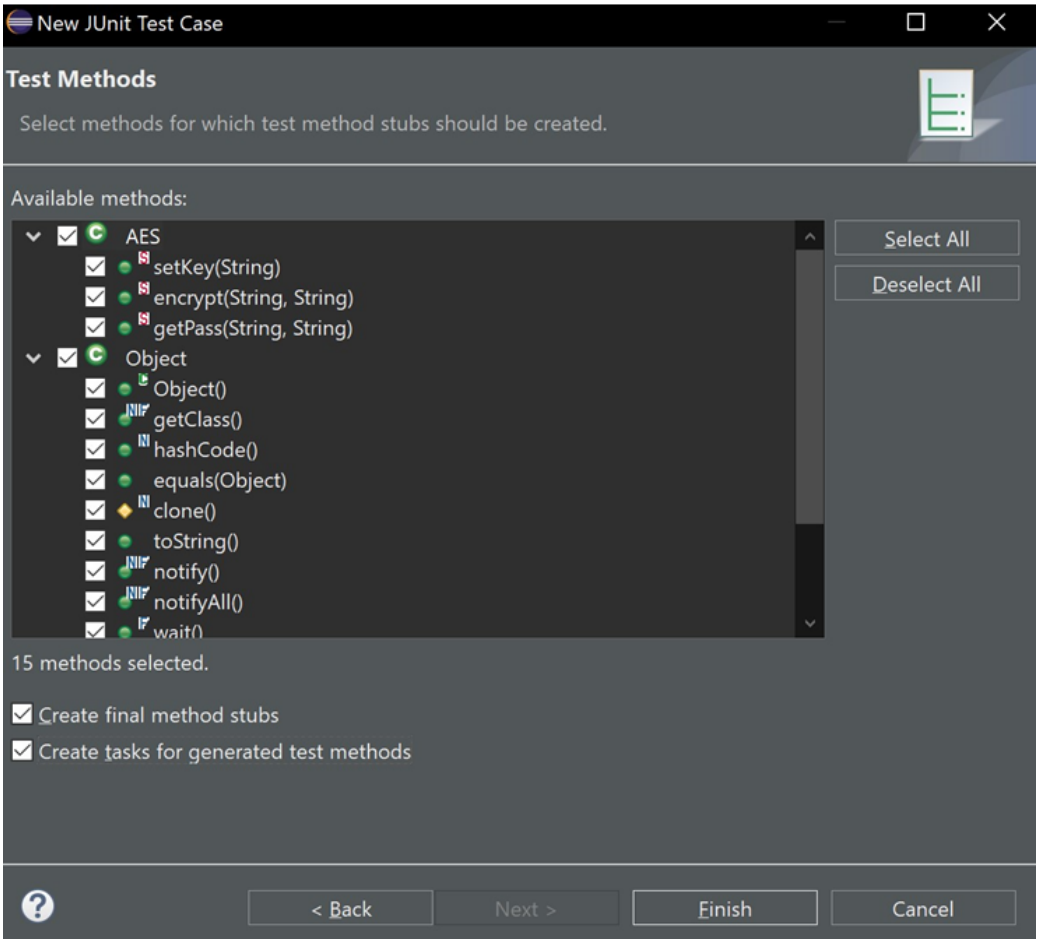
Dynamic Analysis

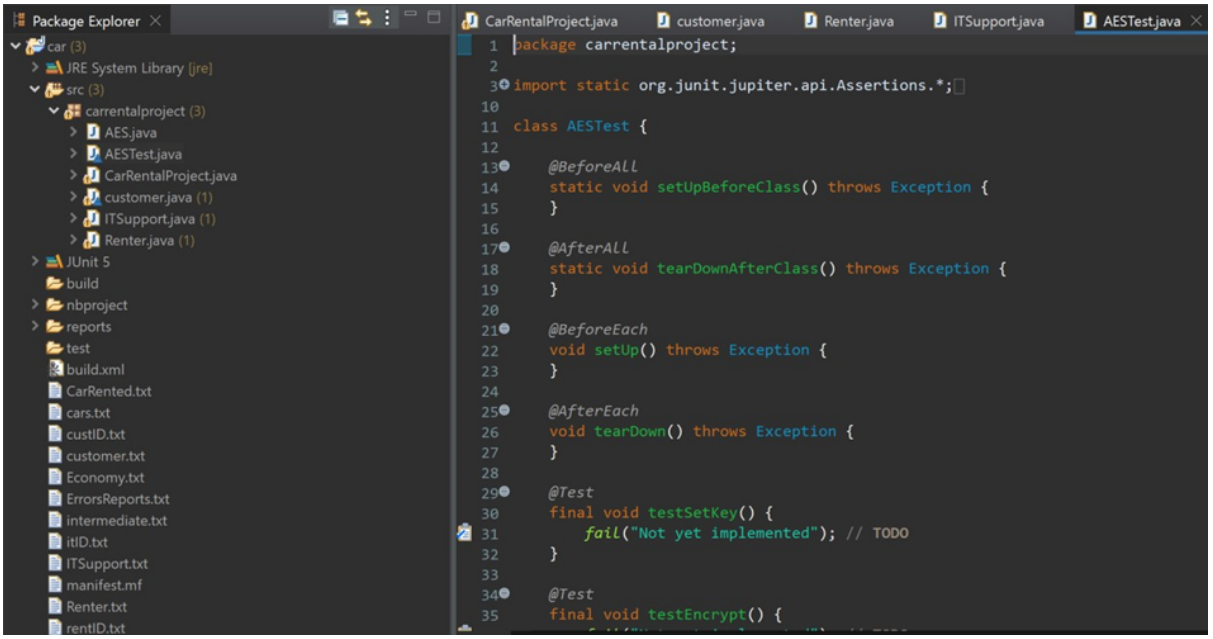
Dynamic testing was performed on the car rental system using the JUnit tool in Eclipse. The focus of the testing was on the AES class, which handles encryption and decryption operations.

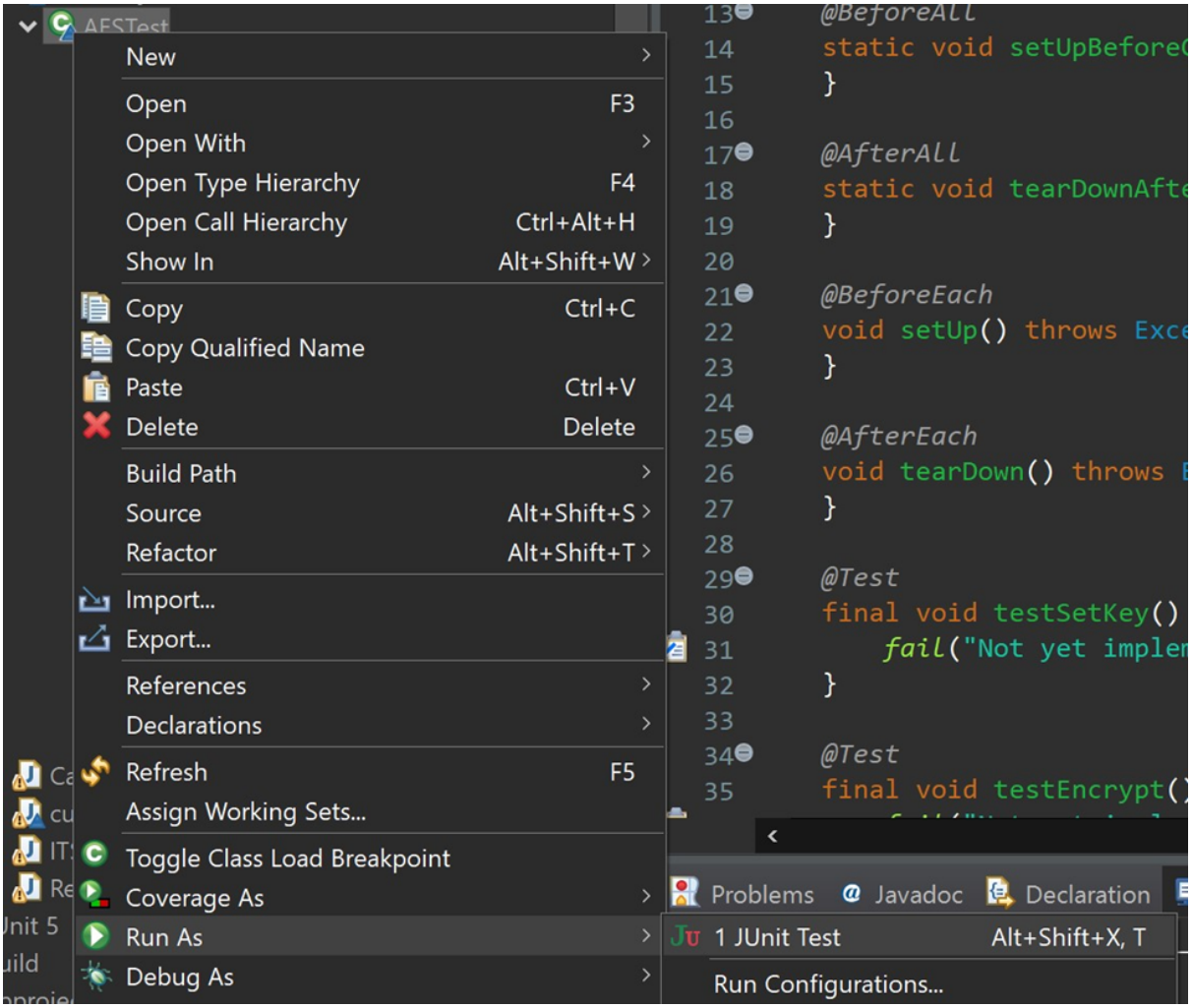
Case 1: AES class

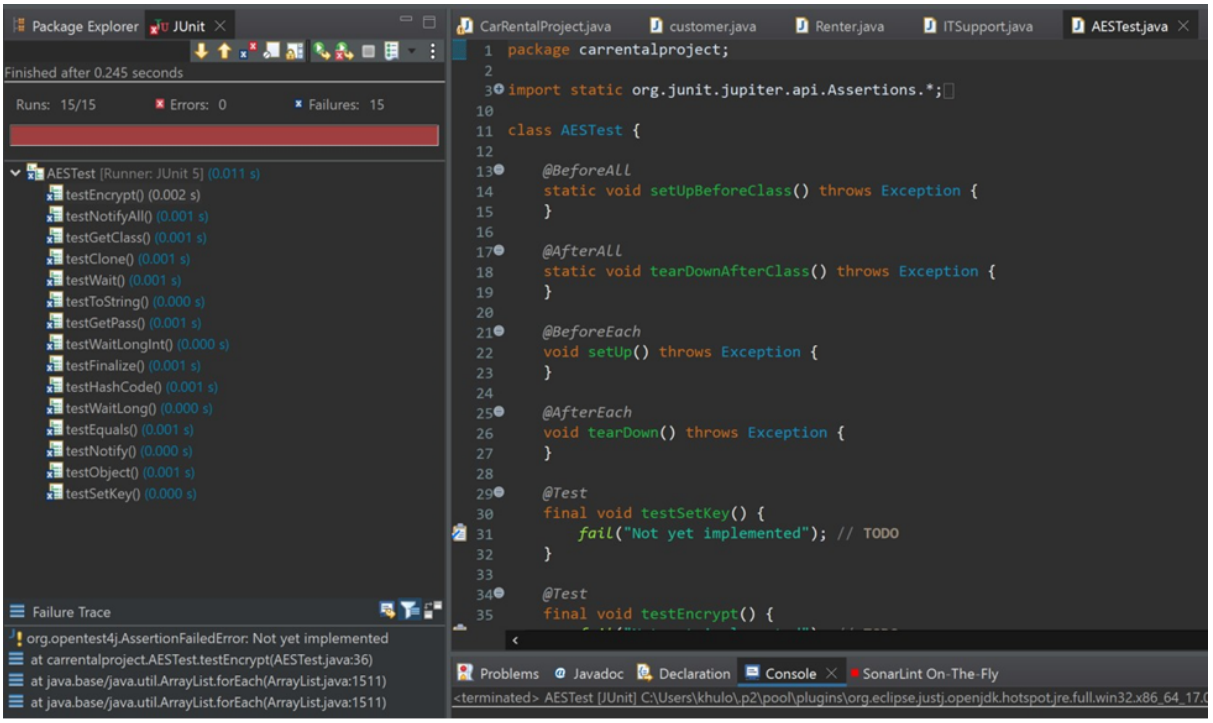

src (3)
carrentalproject (3)
build
nbproject
reports
test
build.xml
cars.txt
custID.txt
ErrorsRep
New
Open F3
Open With
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Java Project
Project...
Package
Class
Interface
Enum
Record
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Other...
Create a JUnit test case
try {
File myObj =
if (myObj.cr

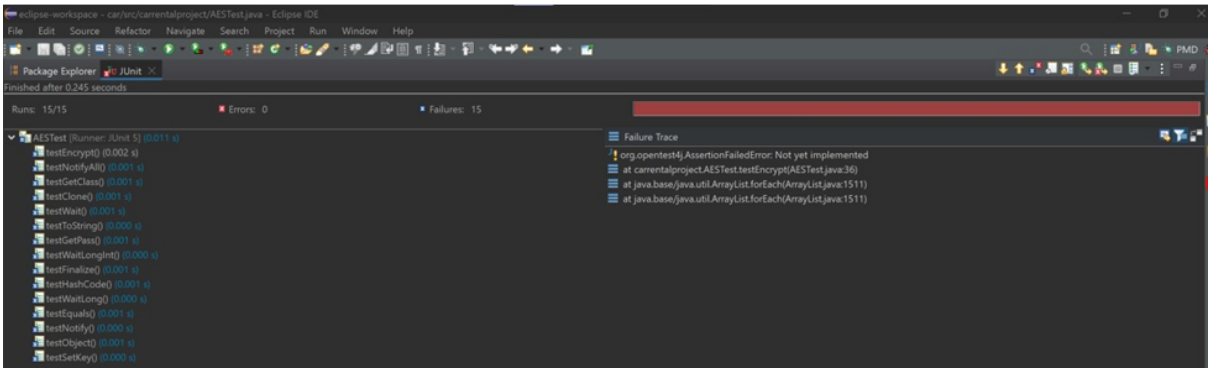

New JUnit Test Case
JUnit Test Case
Select the name of the new JUnit test case. Specify the class under test to select methods to be tested on the next page.
New JUnit 3 test
New JUnit 4 test
New JUnit Jupiter test
Source folder:
car/src
Browse...
Package:
carrentalproject
Browse...
Name:
AESTest
Superclass:
java.lang.Object
Browse...
Which method stubs would you like to create?
@BeforeAll setUpBeforeClass()
@AfterAll tearDownAfterClass()
@BeforeEach setUp()
@AfterEach tearDown()
constructor
Do you want to add comments? (Configure templates and default value here)
Generate comments
Class under test:
carrentalproject.AES
Browse...
< Back
Next >
Finish
Cancel


New JUnit Test Case
Test Methods
Select methods for which test method stubs should be created.
Available methods:
AES
setKey(String)
encrypt(String, String)
getPass(String, String)
Object
Object()
getClass()
hashCode()
equals(Object)
clone()
toString()
notify()
notifyAll()
wait()
Select All
Deselect All
15 methods selected.
Create final method stubs
Create tasks for generated test methods
< Back
Next >
Finish
Cancel


Package Explorer
car (3)
JRE System Library [jre]
src (3)
carrentalproject (3)
AES.java
AESTest.java
CarRentalProject.java
customer.java (1)
ITSupport.java (1)
Renter.java (1)
JUnit 5
build
nbproject
reports
test
build.xml
CarRented.txt
cars.txt
custID.txt
customer.txt
Economy.txt
ErrorsReports.txt
intermediate.txt
itID.txt
ITSupport.txt
manifest.mf
Renter.txt
rentID.txt
CarRentalProject.java
customer.java
Renter.java
ITSupport.java
AESTest.java
1 package carrentalproject;
2
3 import static org.junit.jupiter.api.Assertions.*;
10
11 class AESTest {
12
13 @BeforeAll
14 static void setUpBeforeClass() throws Exception {
15 }
16
17 @AfterAll
18 static void tearDownAfterClass() throws Exception {
19 }
20
21 @BeforeEach
22 void setUp() throws Exception {
23 }
24
25 @AfterEach
26 void tearDown() throws Exception {
27 }
28
29 @Test
30 final void testSetKey() {
31 fail("Not yet implemented"); // TODO
32 }
33
34 @Test
35 final void testEncrypt() {


AESTest
New
Open F3
Open With
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
References
Declarations
Refresh F5
Assign Working Sets...
Toggle Class Load Breakpoint
Coverage As
Run As
Debug As
1 JUnit Test Alt+Shift+X, T
Run Configurations...
Problems
Javadoc
Declaration
@BeforeAll
static void setUpBefore
@AfterAll
static void tearDownAfte
@BeforeEach
void setUp() throws Exce
@AfterEach
void tearDown() throws
@Test
final void testSetKey()
fail("Not yet imple
@Test
final void testEncrypt()



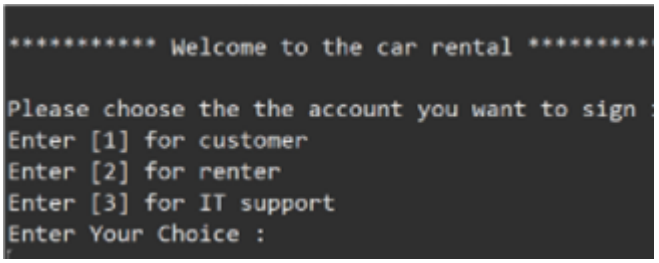


Findings

During dynamic testing of the car rental system, the AES class, responsible for encryption and decryption, exhibited failures. These failures indicate potential implementation issues, such as incorrect algorithms or inadequate input validation. Further investigation is required to identify and address the root causes of the failures, ensuring the AES class functions correctly and provides reliable encryption functionality.

Black Box Testing

Case 1:



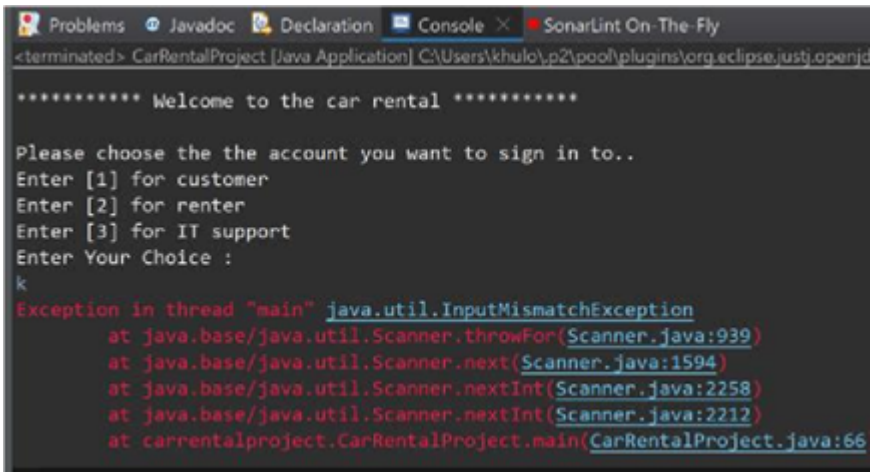
Entering a character as a choice

I choose this case because a character as input shouldn't be accepted as a choice for the menu displayed according to the specification

Expected Results:

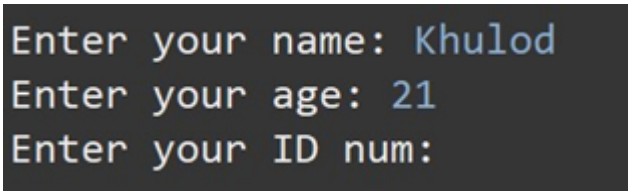
It should display a message that states "you should only enter numbers"

Actual Results:



The character input wasn't handled by the code programmers

Case 2:



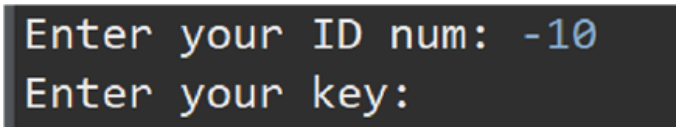
Entering a negative number as an input

I choose this case because a negative number as input shouldn’t be accepted as an ID because there is no valid ID that has characters according to the specification

Expected Results:

It should display a message that states ”you should only enter positive numbers”

Actual Results:



It accepted the negative input

Conclusion

In summary, I performed static analysis using SpotBugs and identified a bug related to redundant encryption. For dynamic testing, I used JUnit to evaluate the system's behavior, uncovering failures in the AES class. Additionally, two black box test cases focused on input validation. These efforts aim to enhance the system's quality, security, and robustness.