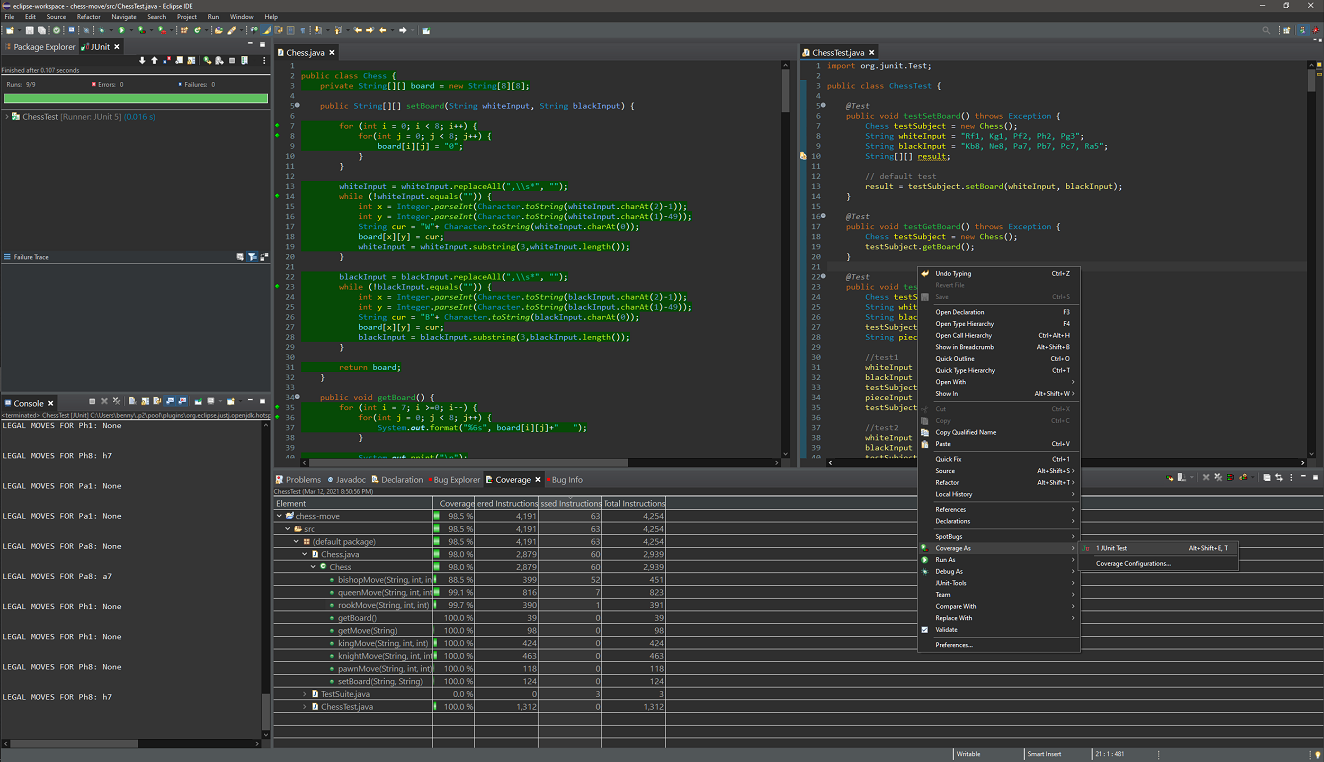
**Manual:**

To run this program, go to Eclipse and open **Chess.java**, **ChessTest.java** and **Main.java** under the same package. Go go ChessTest.java and right click the file and choose **Coverage as -> JUnit Test** as Figure 1 shows, in the Console window you will see the output for each test case, in Coverage you will see the code coverage information. You can run **Main.java** as a Java Application as well, it runs the example in the assignment: it takes input and output the result as the instructions in the assignment document. The *whiteInput*, *blackInput* and *pieceInput* String variables take board configuration from the user input. The *setBoard()*, *getBoard()* and *getMove()* methods perform outputs and calculations.



**Figure 1**

**Test cases:**

**testSetBoard()**: The **testSetBoard()** method tests the board's setup function by accepting configuration from the user.

**testGetBoard()**: The **testGetBoard()** method tests board’s print function.

**testGetMove()**: The **testGetMove()** method tests a piece's possible move output.

**testKingMove()**: The **testKingMove()** method tests all situations of King piece's move, including: move to all directions, partially blocked by friendly pieces, fully blocked, move with take.

**testQueenMove()**: The **testQueenMove()** method tests all situations of Queen piece's move, including: move to all directions, partially blocked by friendly pieces, fully blocked, move with take.

**testRookMove()**: The **testRookMove()** method tests all situations of Rook piece's move, including: move to all directions, partially blocked by friendly pieces, fully blocked, move with take.

**testKnightMove()**: The **testKnightMove()** method tests all situations of Knight piece's move, including: move to all directions, partially blocked by friendly pieces, fully blocked, move with take.

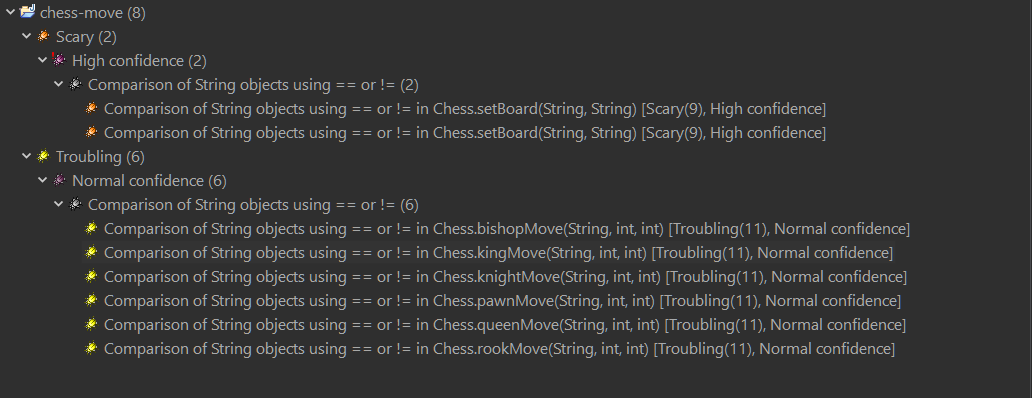
**testBishopMove()**: The **testBishopMove()** method tests all situations of Bishop piece's move, including: move to all directions, partially blocked by friendly pieces, fully blocked, move with take.

**testPawnMove()**: The **testPawnMove()** method tests all situations of Pawn piece's move, including: move to all directions as white, partially blocked by friendly pieces as white, fully blocked as white, move with take as white, move to all directions as black, partially blocked by friendly pieces as black, fully blocked as black, move with take as black

**Static analysis tool improvements:**

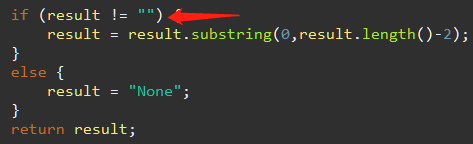
Since FindBugs is not working with the Eclipse Maven plugin, I use SpotBugs instead.

Before I run my program, the SpotBugs shows 8 bugs as Figure 2 shows.

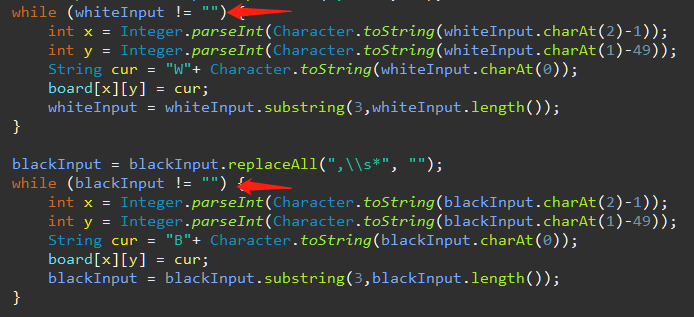


**Figure 2**

I was using == and != to check and compare actual contents of String objects, which is a mistake in Java programming as Figure 3 and 4 shows.

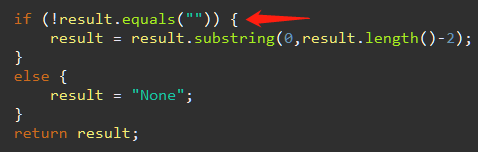


**Figure 3**

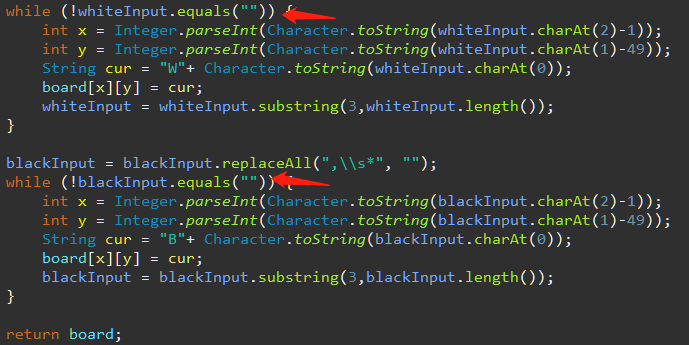


**Figure 4**

Therefore, I changed the comparison method to an appropriate String function: .equals() as Figure 5 and 6 shows.



**Figure 5**

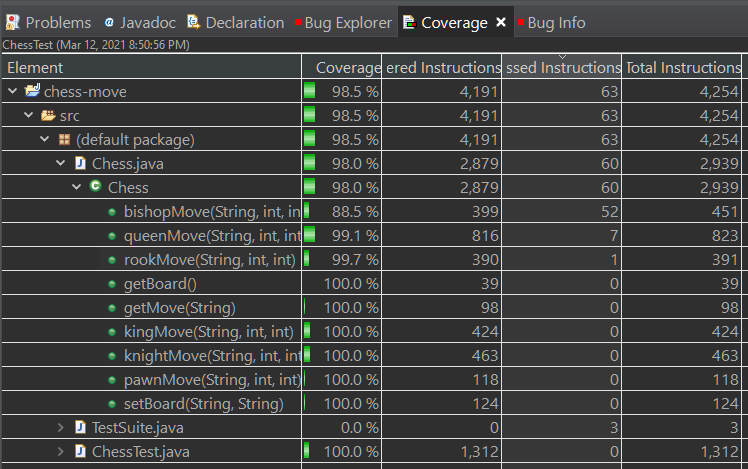


**Figure 6**

After making this modification, the SpotBugs no longer shows anything. Therefore I assume there is no bug in my program.

**Final code coverage report:**

The final code coverage is 98.5% shown as Figure 7. The missing 1.5% is due to the nested **if** statements in move methods of pieces. Although I added all possible situations in test cases, there are conflicts between situations that make certain conditions cannot be triggered while other conditions are satisfied. Therefore the nested **if** statements cannot be covered 100%.



**Figure 7**