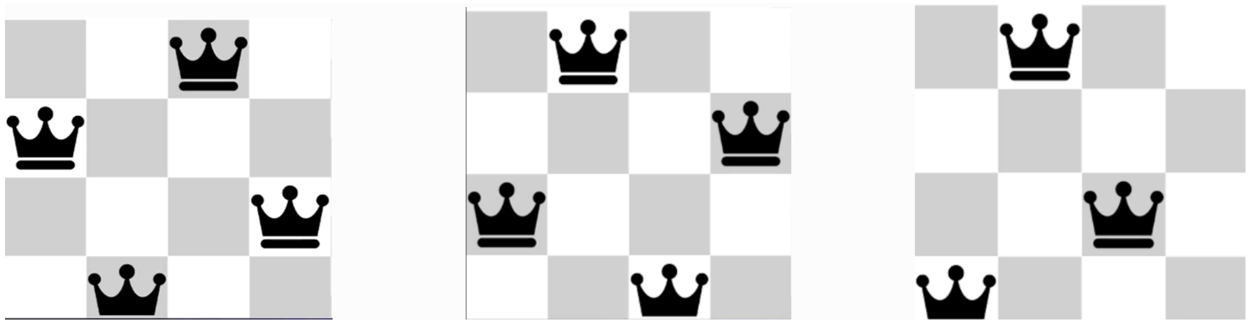
N-QUEEN Problem Documentation

{ **N Queen problem** 👑 }

What is N-Queen problem ?

The *n-queens problem* is to place *n* queens (where *n* > 0) on an *n* -by-*n* chessboard so that no queen is threatened by another one. According to the rules of chess, this is equivalent to the requirement that no two queens be on the same row or the same column or on a common diagonal.

For example , the following is a solution for the 4 Queen problem.



The expected output is in the form of a matrix that has ‘Q’ for the blocks where queens are placed, and the empty spaces are represented by “.” .Foe example ,the following is the output matrix for the above 4-Queen solution.

Input n = 4

Output [[“.Q..”,”...Q”,”Q...”,”..Q.”], [“..Q.”,”Q...”,”...Q”,”.Q..”]

***Solving N Queen Problem using backtracking :**
By using backtracking we position queens in different columns one by one, beginning with the leftmost column. When we position a queen in a column, we look for clashes with other queens that have already been placed. If we locate a row in the current column with no collision, we mark this row and column as part of the solution. We backtrack and return false if we cannot discover such a row due to clashes.
Algorithm: Here’s an algorithm to solve the n queen problem:

- Start with an empty chessboard of size n x n.
- Place the first queen in the top-left corner of the board (i.e., row 1, column 1).

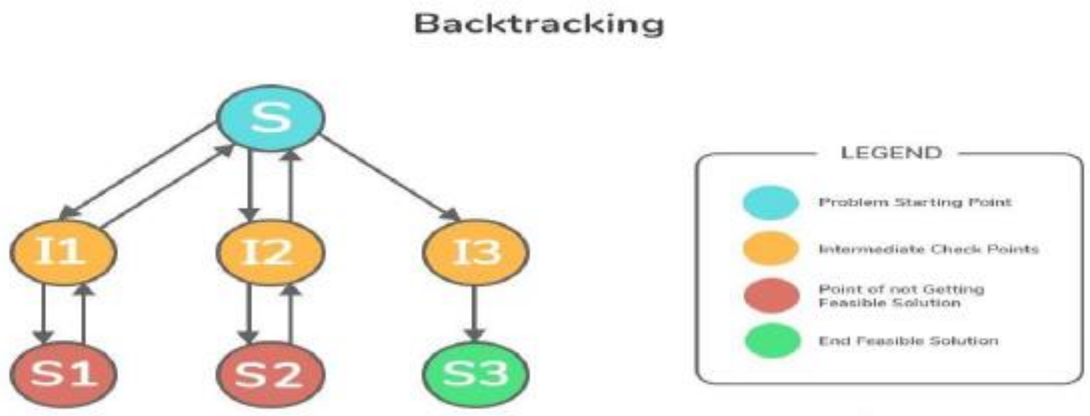
• Move to the next row and try to place a queen in each column until a valid position is found or all columns have been tried.

• If a valid position is found, move to the next row and repeat step 3.

• If all columns have been tried in the current row without finding a valid position, backtrack to the previous row and move the queen to the next column in that row. Repeat step 3.

• If all rows have been tried without finding a solution, backtrack to the previous row and move the queen to the next column in that row. Repeat step 3.

• If a solution is found, print the positions of the queens on the board and exit the program.

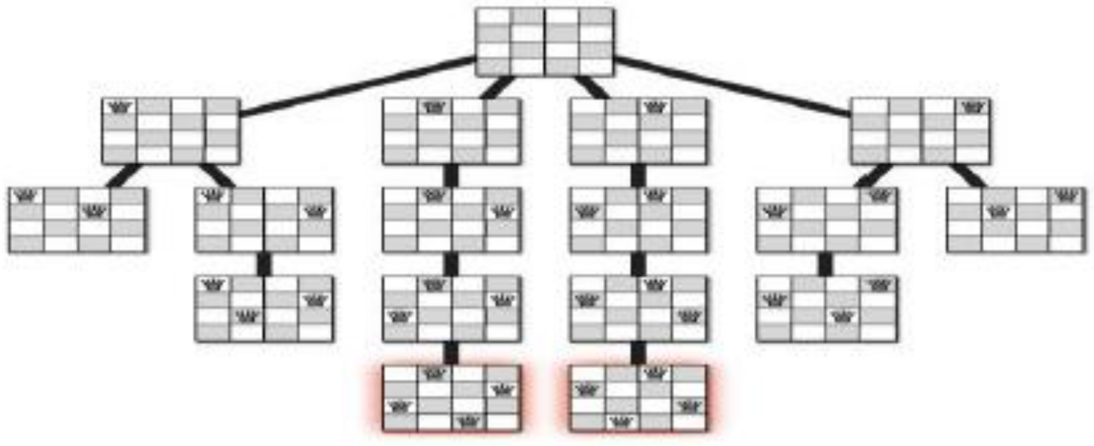


N-Queens Problem In the N-Queens problem, the goal is to place N queens on an N×N chessboard in such a way that no two queens threaten each other. Backtracking efficiently explores possible queen placements.

***Multithreading in the context of solving :**

the N-Queens problem involves the concurrent execution of multiple threads to explore and search for solutions to the problem. The N-Queens problem is a classic puzzle that asks for all possible arrangements of N queens on an N×N chessboard, such that no two queens threaten each other. When using multithreading to solve the N-Queens problem, the goal is to parallelize the search space exploration, allowing multiple threads to work simultaneously on different branches of the solution space. This can lead to improved

performance, especially on multi-core processors, as the workload is distributed across multiple threads.



Volatile Keyword in Java:

The volatile keyword in Java is a crucial tool for writing multi-threaded applications. It ensures thread safety and memory visibility, preventing data races and inconsistent behavior. the core concepts of volatile, its practical usage scenarios, and best practices for effective implementation.

Understanding Volatile: Visibility and Ordering

In a multi-threaded environment, threads can access and modify shared variables concurrently. This can lead to data races, where the outcome depends on the unpredictable timing of thread execution. The volatile keyword tackles this issue by addressing two key aspects:

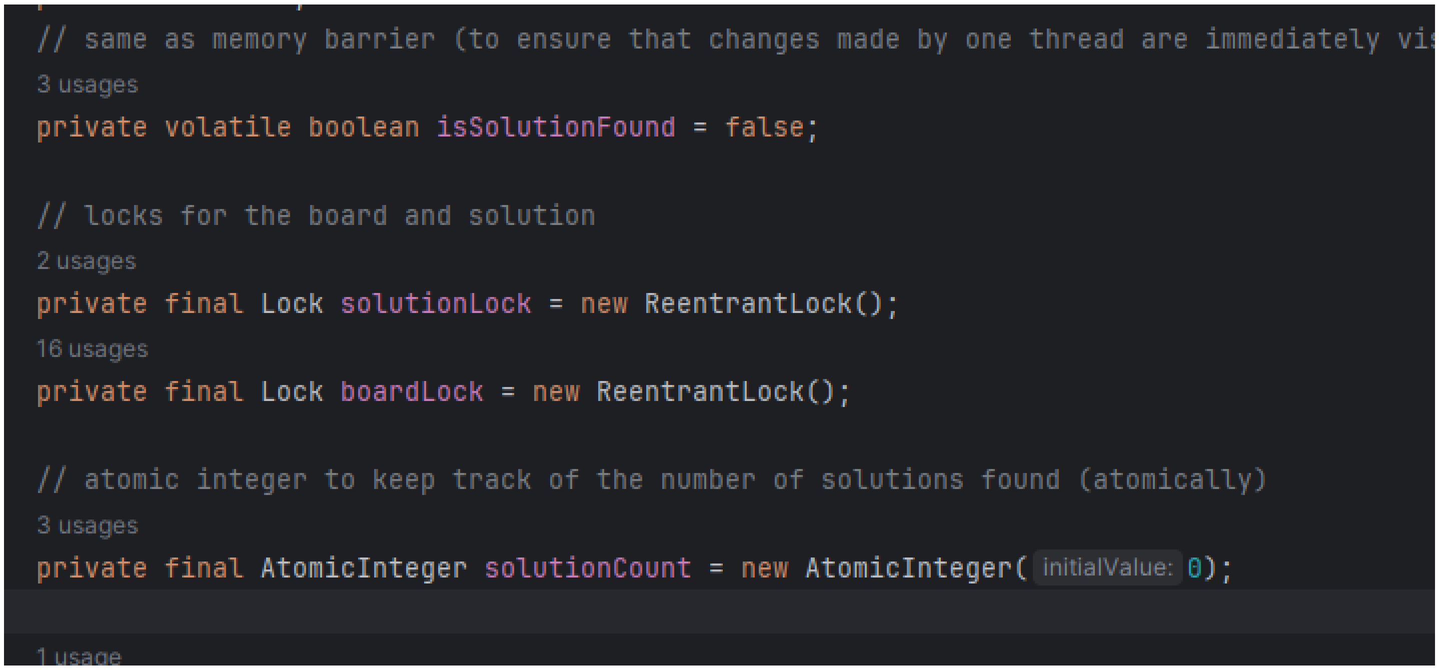
Visibility: When a thread modifies a volatile variable, the change is immediately visible to all other threads. They no longer rely on cached values in their local registers, guaranteeing consistent data access.

Ordering: volatile enforces a memory ordering for read and write operations. This ensures that threads see the changes in the expected order, preventing unexpected behavior like stale data or incorrect synchronization.

***java.util.concurrent.atomic.AtomicInteger:**

class provides operations on underlying int value that can be read and written atomically, and also contains advanced atomic operations. AtomicInteger supports atomic operations on underlying int variable. It have get and set

methods that work like reads and writes on volatile variables. That is, a set has a happens-before relationship with any subsequent get on the same variable. The atomic compareAndSet method also has these memory consistency features.

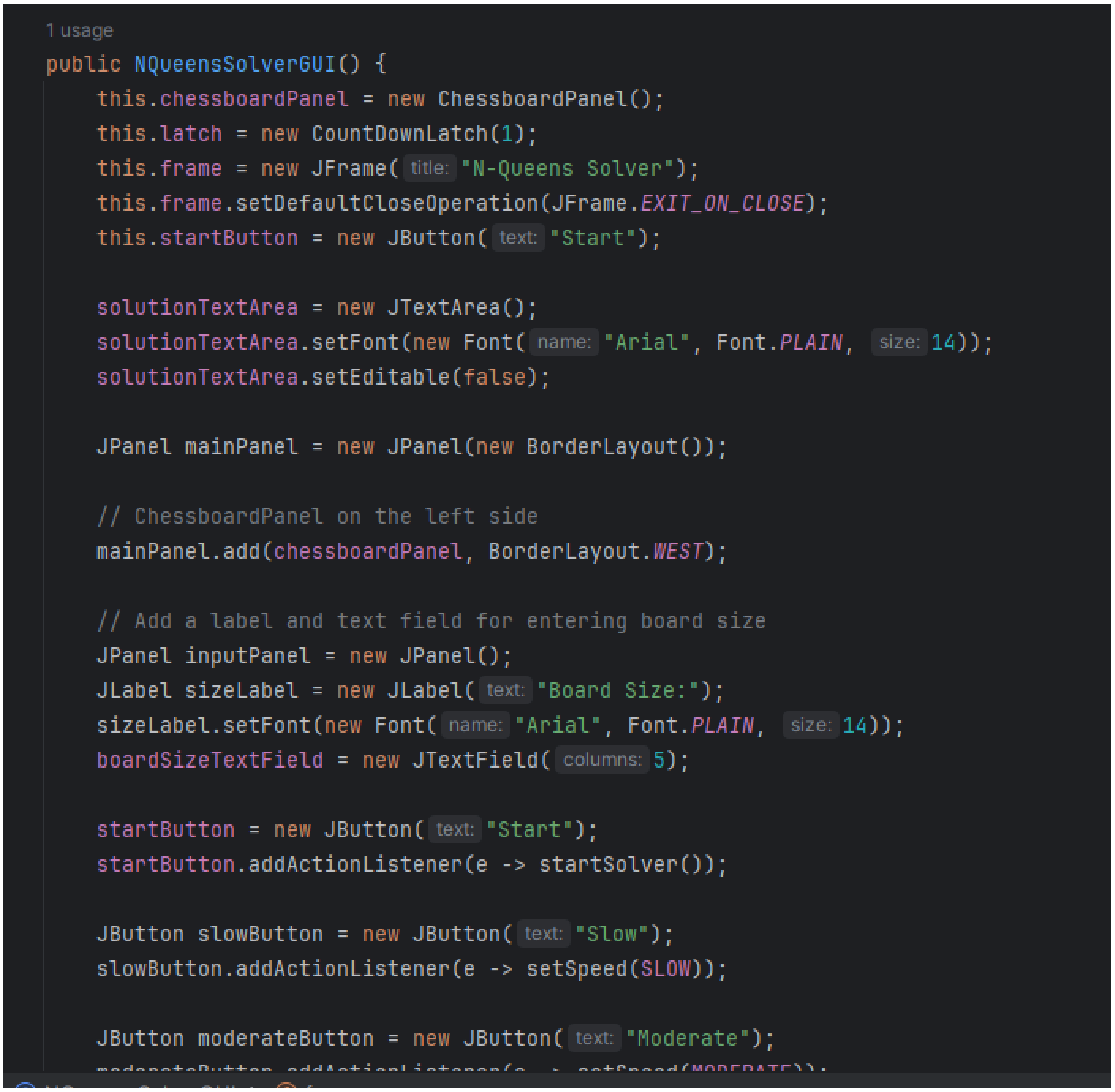


```
// same as memory barrier (to ensure that changes made by one thread are immediately vi
3 usages
private volatile boolean isSolutionFound = false;

// locks for the board and solution
2 usages
private final Lock solutionLock = new ReentrantLock();
16 usages
private final Lock boardLock = new ReentrantLock();

// atomic integer to keep track of the number of solutions found (atomically)
3 usages
private final AtomicInteger solutionCount = new AtomicInteger( initialValue: 0);

1 usage
```

Integration of Java Swing in the N Queen Project

To integrate Java Swing into the N Queen project, follow these steps:

1. **Create a Swing Frame:** Start by creating a JFrame object, which serves as the main window for the N Queen application. The frame provides a container to hold various Swing components.
2. **Design the User Interface:** Use Swing components like buttons, labels, and panels to design the GUI for the N Queen application. Determine the layout managers that best suit the requirements and arrange the components accordingly.
3. **Implement Event Handling:** Attach event listeners to the relevant Swing components to capture user interactions. For example, you can associate a button with an event listener to handle the click event and perform specific actions when the button is clicked.
4. **Integrate with N Queen Logic:** Connect the Swing components with the existing N Queen logic. For instance, when the user interacts with the GUI, invoke the corresponding N Queen methods to update the board state and display the results.
5. **Compile and Run:** Compile the Java source code, including the Swing components, and execute the N Queen application. The Swing GUI should now be displayed, allowing users to interact with the N Queen solver.



```
1 usage
public NQueensSolverGUI() {
    this.chessboardPanel = new ChessboardPanel();
    this.latch = new CountDownLatch(1);
    this.frame = new JFrame( title: "N-Queens Solver");
    this.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.startButton = new JButton( text: "Start");

    solutionTextArea = new JTextArea();
    solutionTextArea.setFont(new Font( name: "Arial", Font.PLAIN,  size: 14));
    solutionTextArea.setEditable(false);

    JPanel mainPanel = new JPanel(new BorderLayout());

    // ChessboardPanel on the left side
    mainPanel.add(chessboardPanel, BorderLayout.WEST);

    // Add a label and text field for entering board size
    JPanel inputPanel = new JPanel();
    JLabel sizeLabel = new JLabel( text: "Board Size:");
    sizeLabel.setFont(new Font( name: "Arial", Font.PLAIN,  size: 14));
    boardSizeTextField = new JTextField( columns: 5);

    startButton = new JButton( text: "Start");
    startButton.addActionListener(e -> startSolver());

    JButton slowButton = new JButton( text: "Slow");
    slowButton.addActionListener(e -> setSpeed(SLOW));

    JButton moderateButton = new JButton( text: "Moderate");
    moderateButton.addActionListener(e -> setSpeed(MODERATE));
```