جامعة حلوان
HELWAN UNIVERSITY

Operating Systems 2 (Fall 2023)
Project Discussion

كلية الحاسبات والذكاء الإصطناعي
Faculty of Computers & Artificial Intelligence

Operating systems 2 project 4 documentation

Project description:

The N-Queens Solver is a Java application that aims to find solutions to the N-Queens problem. The N-Queens problem involves placing N queens on an N×N chessboard in such a way that no two queens threaten each other. The project utilizes a multithreaded approach for solving the problem and includes a graphical user interface (GUI) to visualize the solutions.

Example :

Input : n = 4

Output : there will be two distinct solutions to the 4-queens puzzle as shown below

```
1) 0 1 0 0
   0 0 0 1
   1 0 0 0
   0 0 1 0

2) 0 0 1 0
   1 0 0 0
   0 0 0 1
   0 1 0 0
```

What we have actually did:

Project Overview

- **NQueensSolver.java**: Implements the core logic for solving the N-Queens problem.
- **NQueensThread.java**: Represents a thread for solving the problem concurrently.
- **NQueensSolverGUI.java**: Provides a graphical user interface for interacting with the solver.
- **Main.java**: The main entry point for running the application.
- **pom.xml**: Maven configuration for managing dependencies.

Project Structure

The project follows a modular structure:

- com.example.nqueenssolver
  - Main.java
  - solver
    - NQueensSolver.java
    - NQueensThread.java
  - gui
    - NQueensSolverGUI.java

Algorithm Details

The algorithm uses a backtracking approach to explore all possible combinations of queen placements. The time complexity is (O(N!)), and the space complexity is (O(N)).

Concurrency

The project utilizes multithreading to solve the N-Queens problem concurrently. Each thread represents a unique attempt to find a solution.

Graphical User Interface (GUI)

The GUI, implemented using Java Swing, allows users to input the size of the chessboard (N) and visualize the solutions found by each thread.

Dependencies

- Java Swing for GUI components.
- Maven for project management.

Performance Considerations

The actual runtime may vary based on the hardware and environment. Consider adjusting thread count and other parameters for optimal performance.

Team members roles:

Ahmed Fatthi (Team Leader) : Core logic (backtracking algorithm),

GUI, Documentation and Multithreading.

Ahmed Fadel : Testing , and documentation.

Osama Eid : Testing , and Core Logic (backtracking algorithm)

Ahmed Moshrif : GUI and Documentation.

Mazen Essam : GUI and Testing.

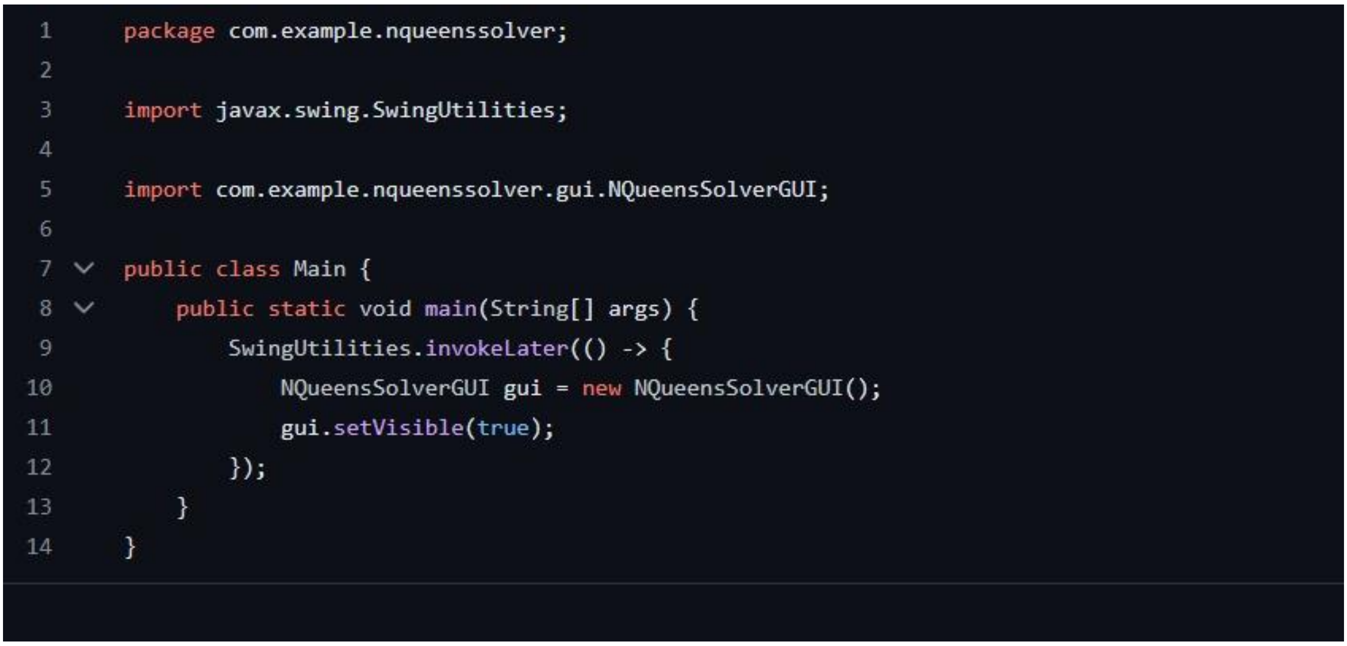
Mahmoud Gamal : Multithreading and Testing.

Ismail Mohamed : Documentation and GUI.

Code documentation:

Main.java:

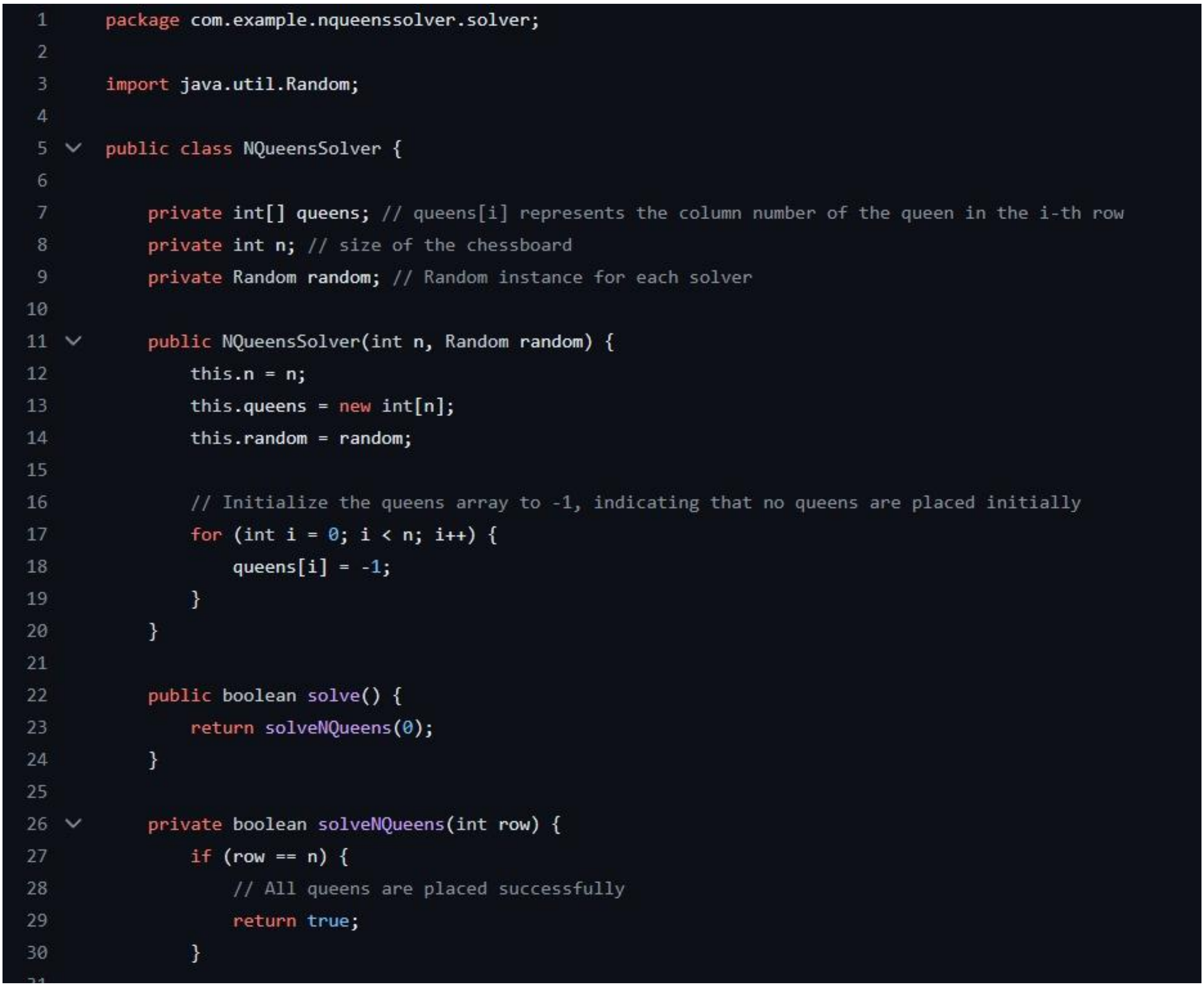
This code initializes and displays the GUI for an N-Queens solver in a Swing application. The actual implementation of the solver logic and GUI components would be contained within the `NQueensSolverGUI` class.



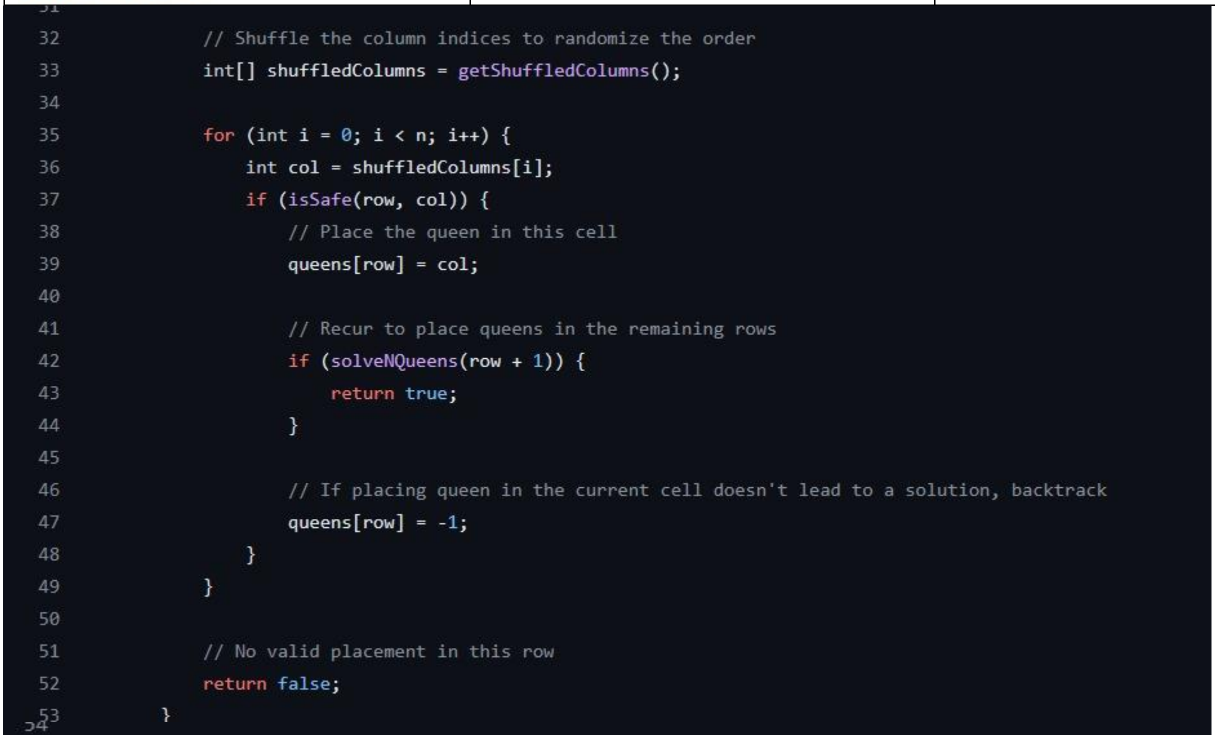
```java
package com.example.nqueenssolver;

import javax.swing.SwingUtilities;

import com.example.nqueenssolver.gui.NQueensSolverGUI;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            NQueensSolverGUI gui = new NQueensSolverGUI();
            gui.setVisible(true);
        });
    }
}
```

NQueensSolver.java:

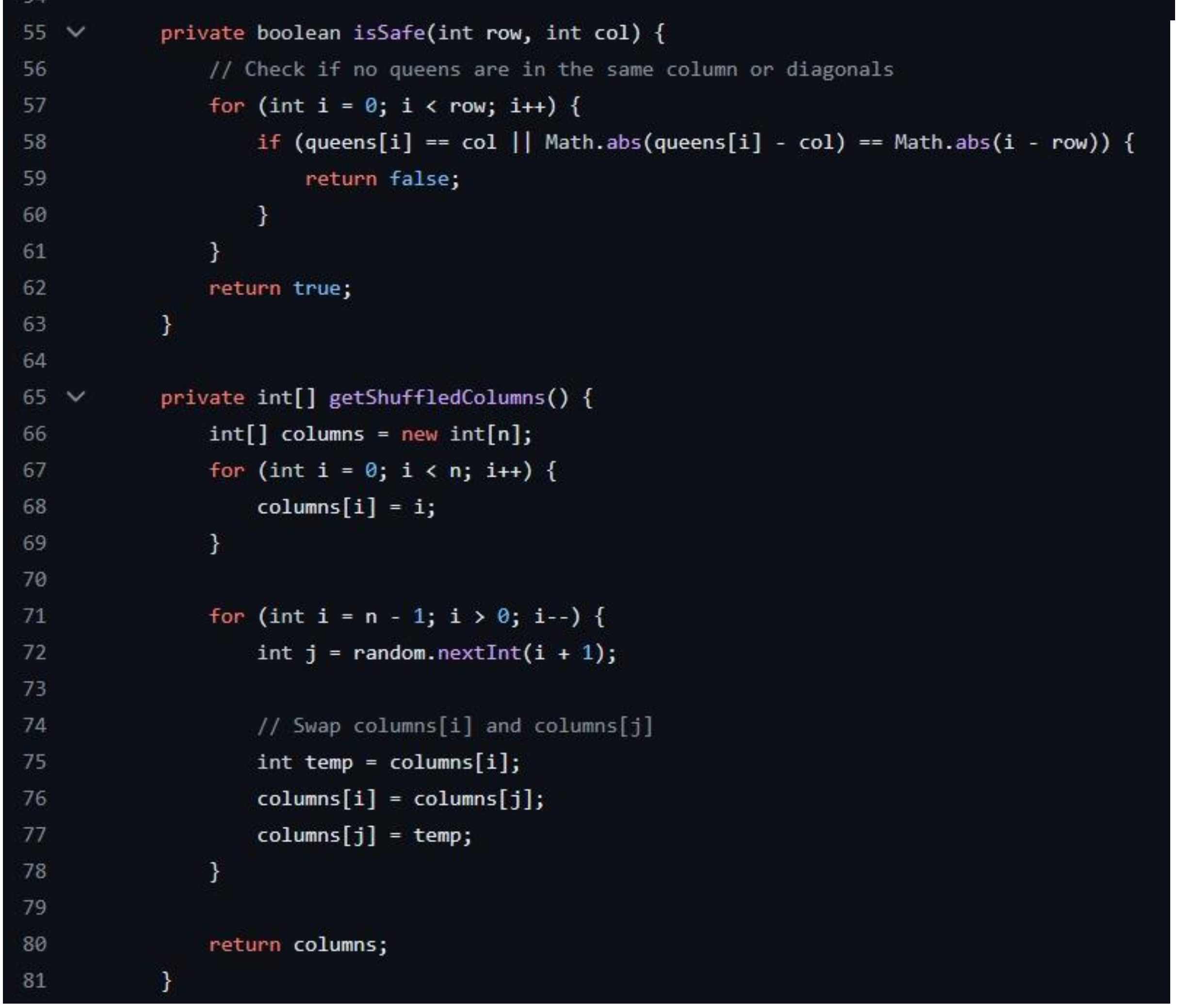
This code provides a flexible and randomized solver for the N-Queens problem, allowing for methods to get the current state of the chessboard, place and remove queens, and retrieve the final solution. The solver employs backtracking and randomness to find a solution.



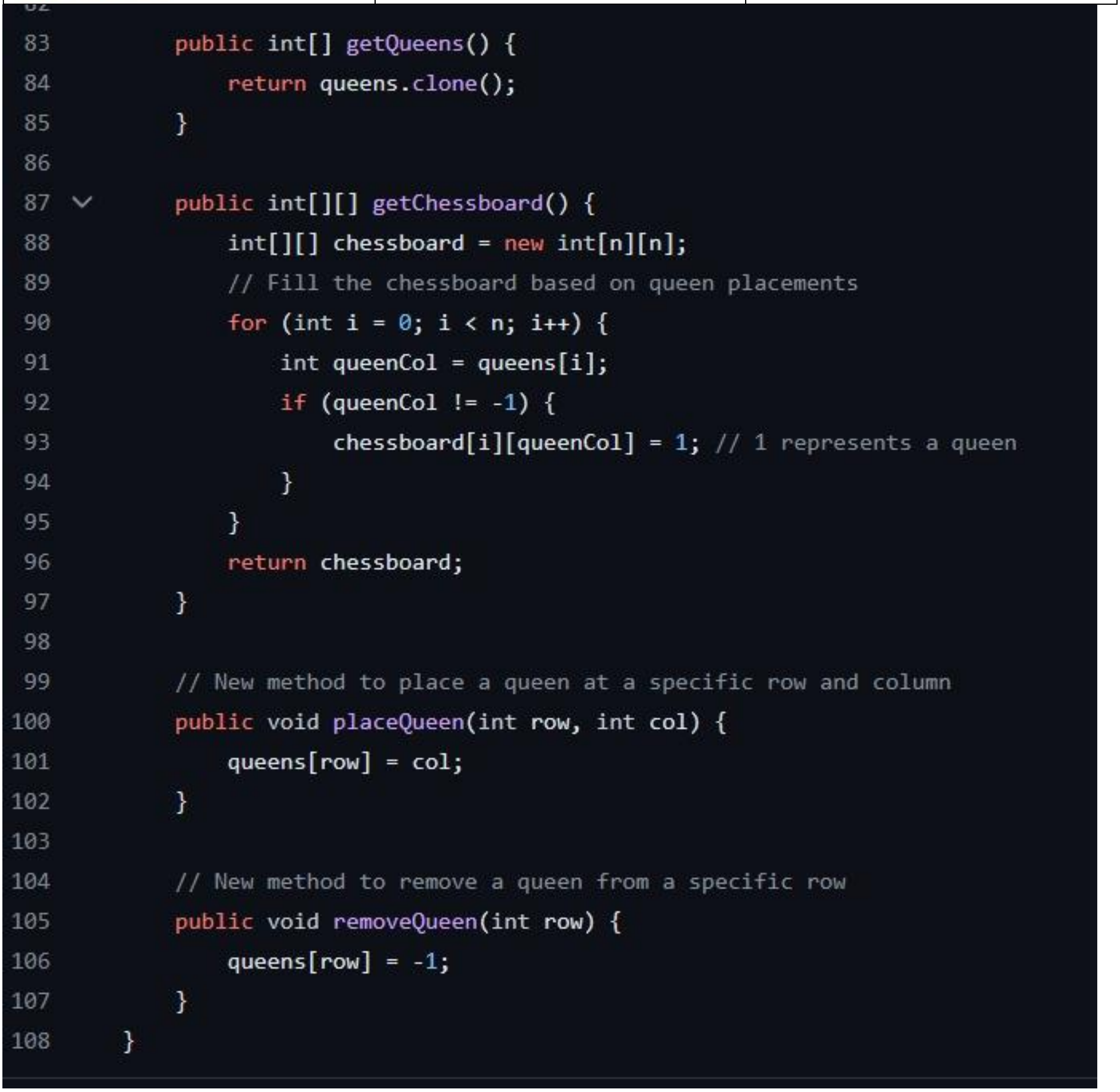
```java
package com.example.nqueenssolver.solver;

import java.util.Random;

public class NQueensSolver {

    private int[] queens; // queens[i] represents the column number of the queen in the i-th row
    private int n; // size of the chessboard
    private Random random; // Random instance for each solver

    public NQueensSolver(int n, Random random) {
        this.n = n;
        this.queens = new int[n];
        this.random = random;

        // Initialize the queens array to -1, indicating that no queens are placed initially
        for (int i = 0; i < n; i++) {
            queens[i] = -1;
        }
    }

    public boolean solve() {
        return solveNQueens(0);
    }

    private boolean solveNQueens(int row) {
        if (row == n) {
            // All queens are placed successfully
            return true;
        }
```



```java
        // Shuffle the column indices to randomize the order
        int[] shuffledColumns = getShuffledColumns();

        for (int i = 0; i < n; i++) {
            int col = shuffledColumns[i];
            if (isSafe(row, col)) {
                // Place the queen in this cell
                queens[row] = col;

                // Recur to place queens in the remaining rows
                if (solveNQueens(row + 1)) {
                    return true;
                }

                // If placing queen in the current cell doesn't lead to a solution, backtrack
                queens[row] = -1;
            }
        }

        // No valid placement in this row
        return false;
    }
```



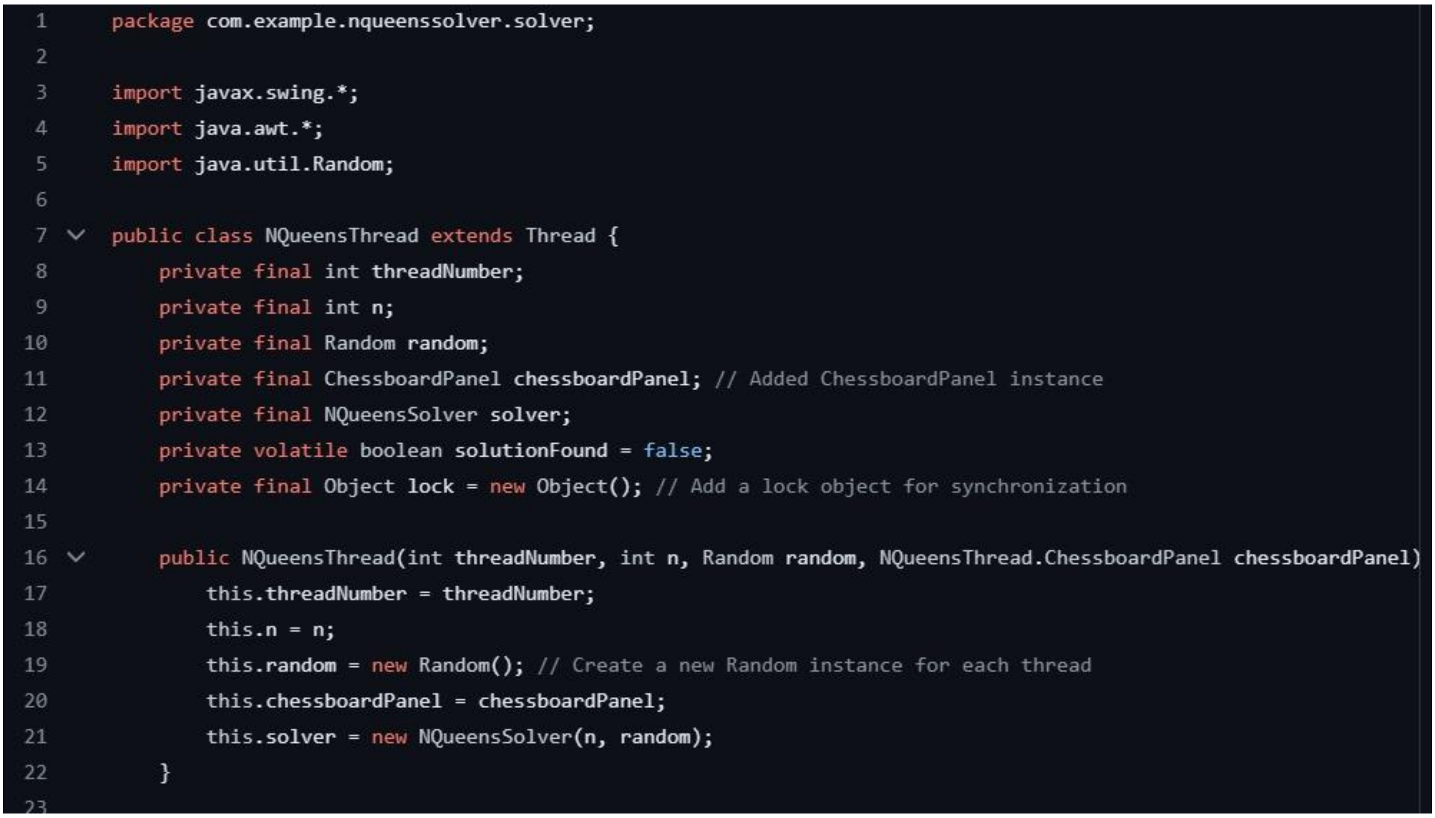
```java
    private boolean isSafe(int row, int col) {
        // Check if no queens are in the same column or diagonals
        for (int i = 0; i < row; i++) {
            if (queens[i] == col || Math.abs(queens[i] - col) == Math.abs(i - row)) {
                return false;
            }
        }
        return true;
    }

    private int[] getShuffledColumns() {
        int[] columns = new int[n];
        for (int i = 0; i < n; i++) {
            columns[i] = i;
        }

        for (int i = n - 1; i > 0; i--) {
            int j = random.nextInt(i + 1);

            // Swap columns[i] and columns[j]
            int temp = columns[i];
            columns[i] = columns[j];
            columns[j] = temp;
        }

        return columns;
    }
```



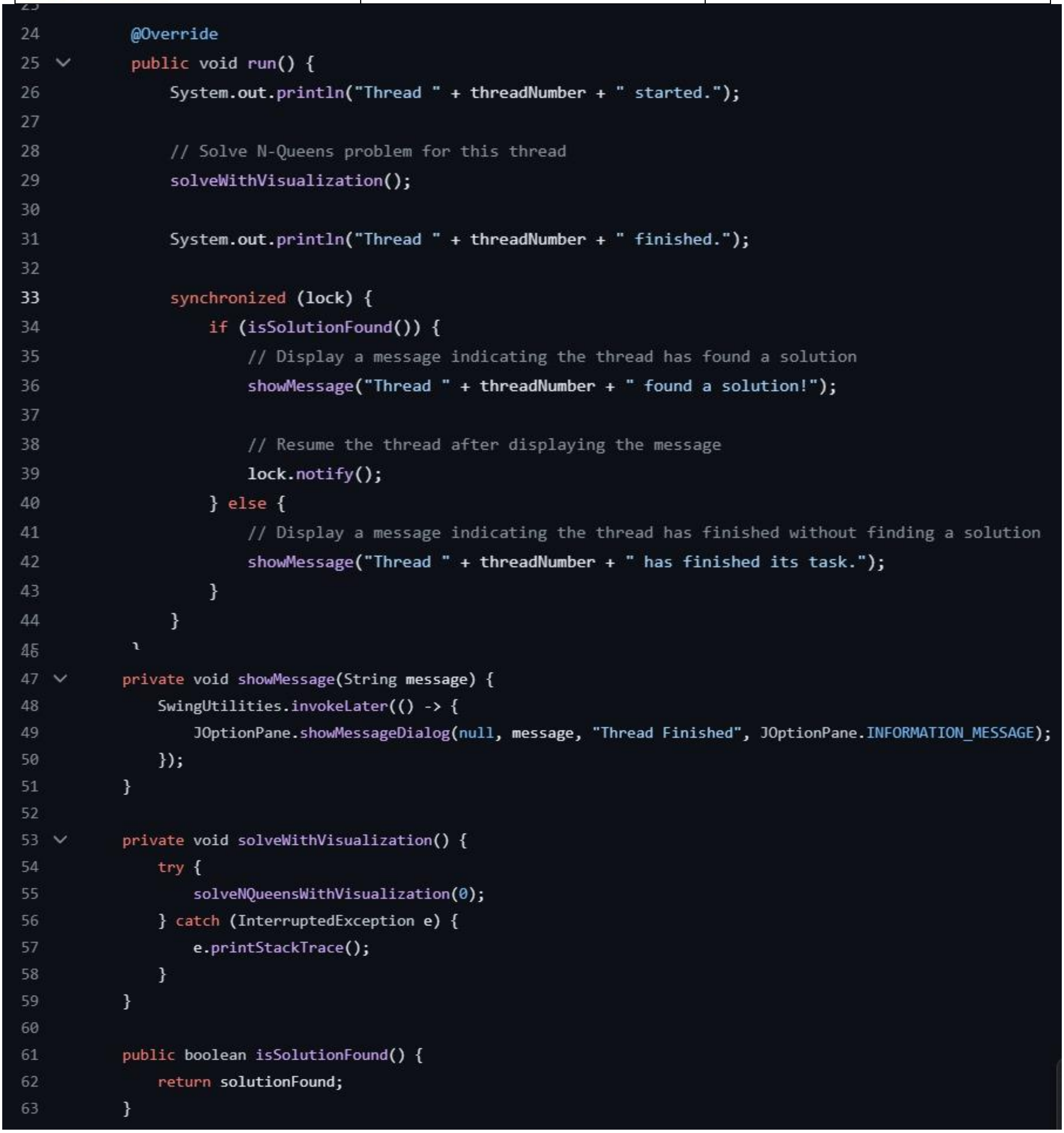
```
    public int[] getQueens() {
        return queens.clone();
    }

    public int[][] getChessboard() {
        int[][] chessboard = new int[n][n];
        // Fill the chessboard based on queen placements
        for (int i = 0; i < n; i++) {
            int queenCol = queens[i];
            if (queenCol != -1) {
                chessboard[i][queenCol] = 1; // 1 represents a queen
            }
        }
        return chessboard;
    }

    // New method to place a queen at a specific row and column
    public void placeQueen(int row, int col) {
        queens[row] = col;
    }

    // New method to remove a queen from a specific row
    public void removeQueen(int row) {
        queens[row] = -1;
    }
}
```

NQueensThread.java:

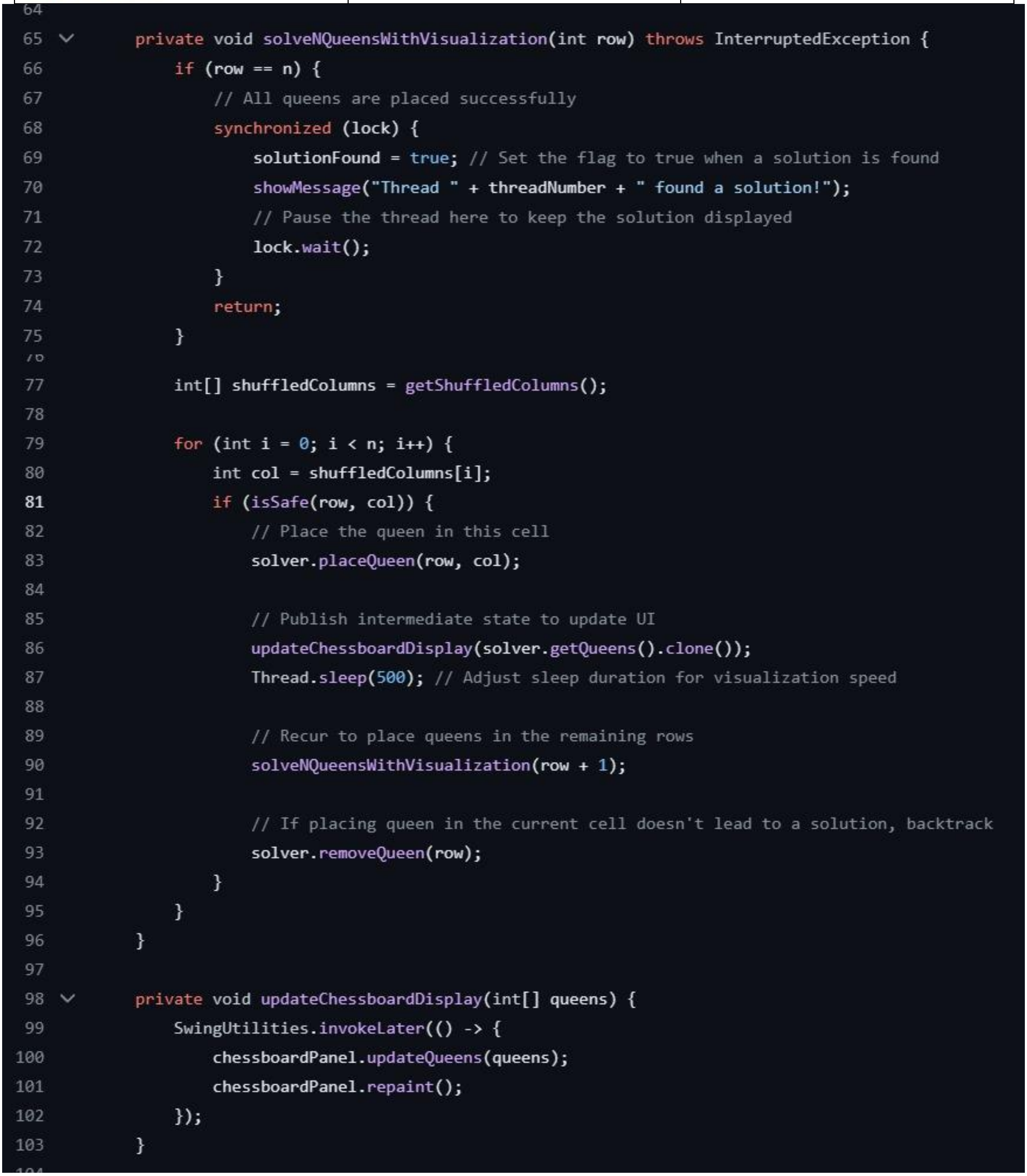
This code represents a threaded version of the N-Queens solver with a graphical visualization of the solving process. Each thread has its own chessboard display, and solutions are displayed with messages using Swing utilities. The `'lock'` object is used for synchronization to control the flow of messages and thread execution.



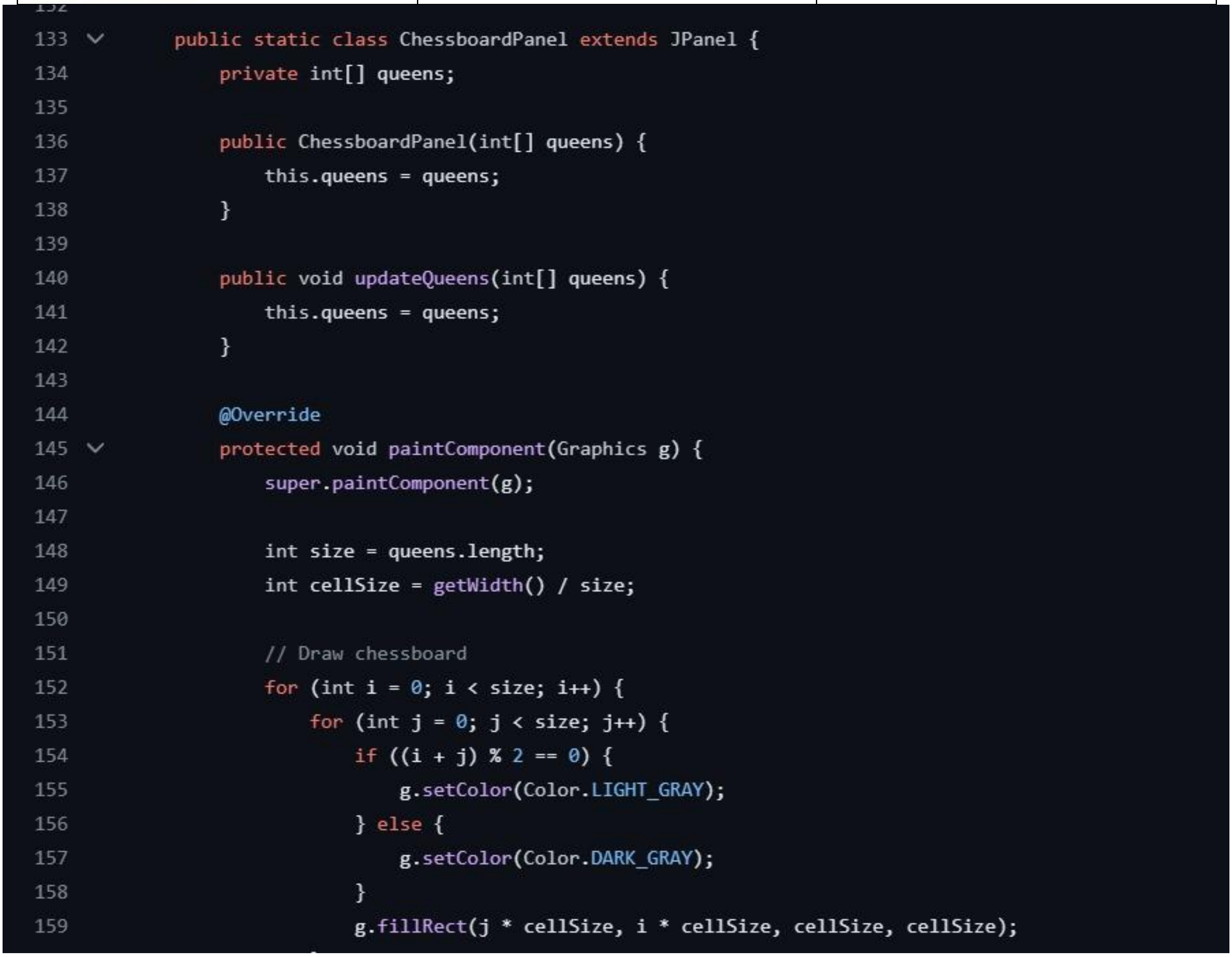
```java
package com.example.nqueenssolver.solver;

import javax.swing.*;
import java.awt.*;
import java.util.Random;

public class NQueensThread extends Thread {
    private final int threadNumber;
    private final int n;
    private final Random random;
    private final ChessboardPanel chessboardPanel; // Added ChessboardPanel instance
    private final NQueensSolver solver;
    private volatile boolean solutionFound = false;
    private final Object lock = new Object(); // Add a lock object for synchronization

    public NQueensThread(int threadNumber, int n, Random random, NQueensThread.ChessboardPanel chessboardPanel)
        this.threadNumber = threadNumber;
        this.n = n;
        this.random = new Random(); // Create a new Random instance for each thread
        this.chessboardPanel = chessboardPanel;
        this.solver = new NQueensSolver(n, random);
    }
```



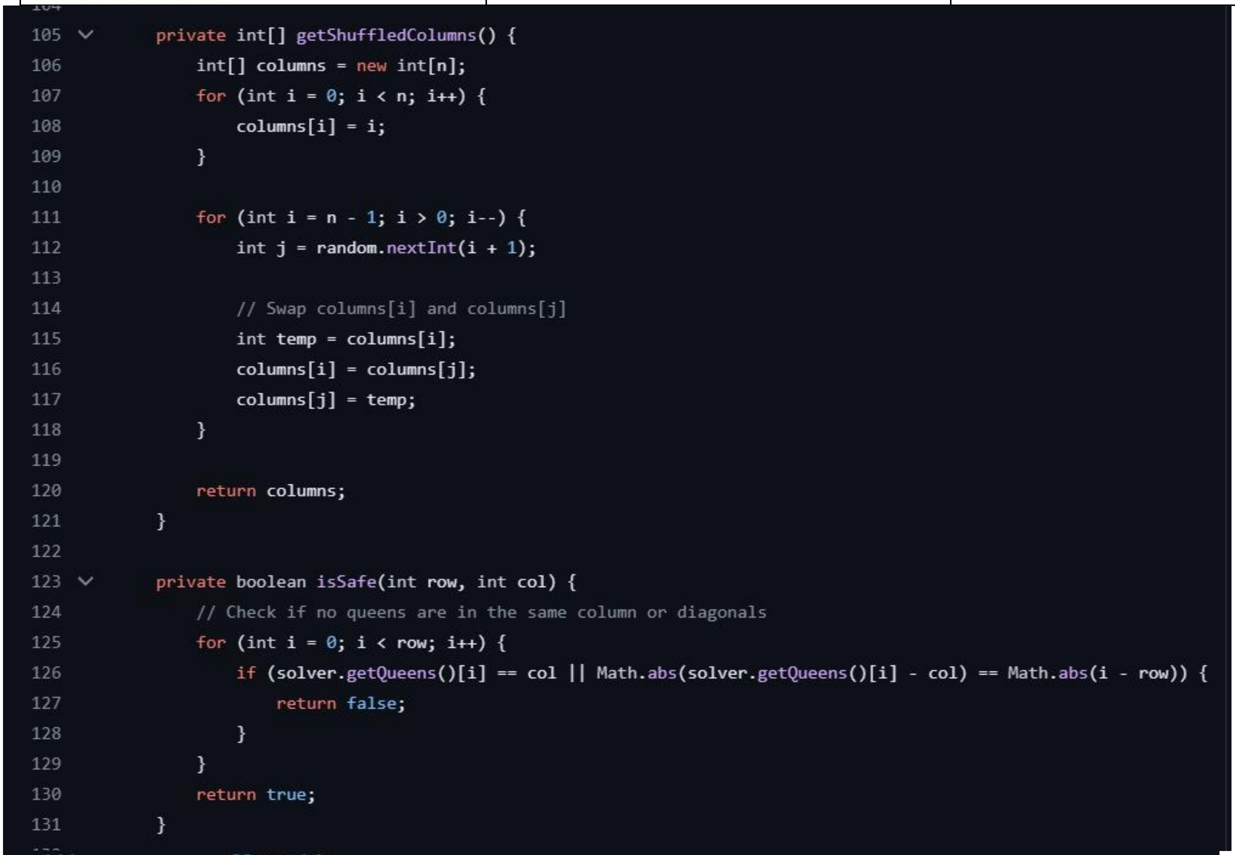
```java
    @Override
    public void run() {
        System.out.println("Thread " + threadNumber + " started.");

        // Solve N-Queens problem for this thread
        solveWithVisualization();

        System.out.println("Thread " + threadNumber + " finished.");

        synchronized (lock) {
            if (isSolutionFound()) {
                // Display a message indicating the thread has found a solution
                showMessage("Thread " + threadNumber + " found a solution!");

                // Resume the thread after displaying the message
                lock.notify();
            } else {
                // Display a message indicating the thread has finished without finding a solution
                showMessage("Thread " + threadNumber + " has finished its task.");
            }
        }
    }
    private void showMessage(String message) {
        SwingUtilities.invokeLater(() -> {
            JOptionPane.showMessageDialog(null, message, "Thread Finished", JOptionPane.INFORMATION_MESSAGE);
        });
    }

    private void solveWithVisualization() {
        try {
            solveNQueensWithVisualization(0);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public boolean isSolutionFound() {
        return solutionFound;
    }
```



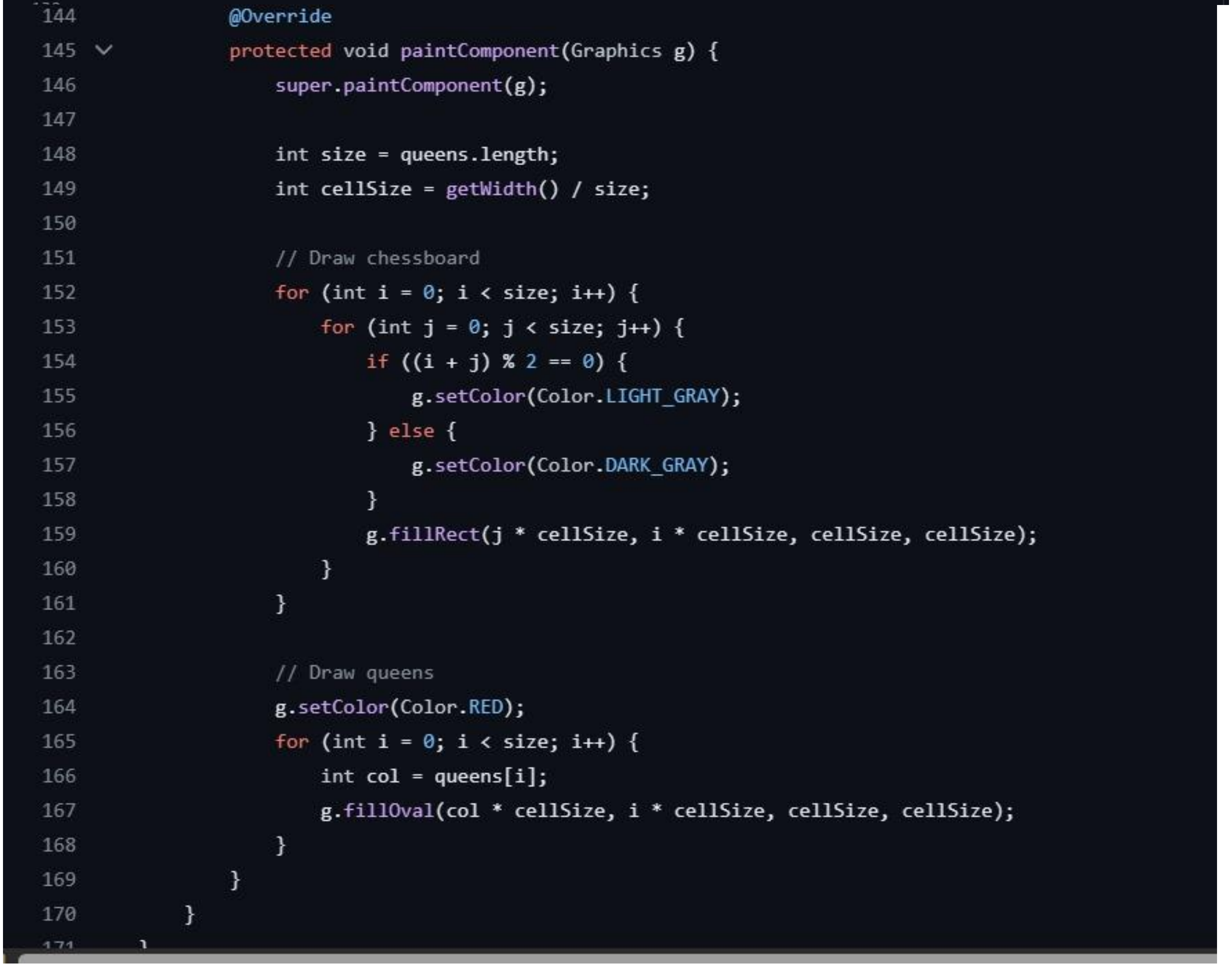
```java
    private void solveNQueensWithVisualization(int row) throws InterruptedException {
        if (row == n) {
            // All queens are placed successfully
            synchronized (lock) {
                solutionFound = true; // Set the flag to true when a solution is found
                showMessage("Thread " + threadNumber + " found a solution!");
                // Pause the thread here to keep the solution displayed
                lock.wait();
            }
            return;
        }

        int[] shuffledColumns = getShuffledColumns();

        for (int i = 0; i < n; i++) {
            int col = shuffledColumns[i];
            if (isSafe(row, col)) {
                // Place the queen in this cell
                solver.placeQueen(row, col);

                // Publish intermediate state to update UI
                updateChessboardDisplay(solver.getQueens().clone());
                Thread.sleep(500); // Adjust sleep duration for visualization speed

                // Recur to place queens in the remaining rows
                solveNQueensWithVisualization(row + 1);

                // If placing queen in the current cell doesn't lead to a solution, backtrack
                solver.removeQueen(row);
            }
        }
    }

    private void updateChessboardDisplay(int[] queens) {
        SwingUtilities.invokeLater(() -> {
            chessboardPanel.updateQueens(queens);
            chessboardPanel.repaint();
        });
    }
```



```java
    public static class ChessboardPanel extends JPanel {
        private int[] queens;

        public ChessboardPanel(int[] queens) {
            this.queens = queens;
        }

        public void updateQueens(int[] queens) {
            this.queens = queens;
        }

        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);

            int size = queens.length;
            int cellSize = getWidth() / size;

            // Draw chessboard
            for (int i = 0; i < size; i++) {
                for (int j = 0; j < size; j++) {
                    if ((i + j) % 2 == 0) {
                        g.setColor(Color.LIGHT_GRAY);
                    } else {
                        g.setColor(Color.DARK_GRAY);
                    }
                    g.fillRect(j * cellSize, i * cellSize, cellSize, cellSize);
```



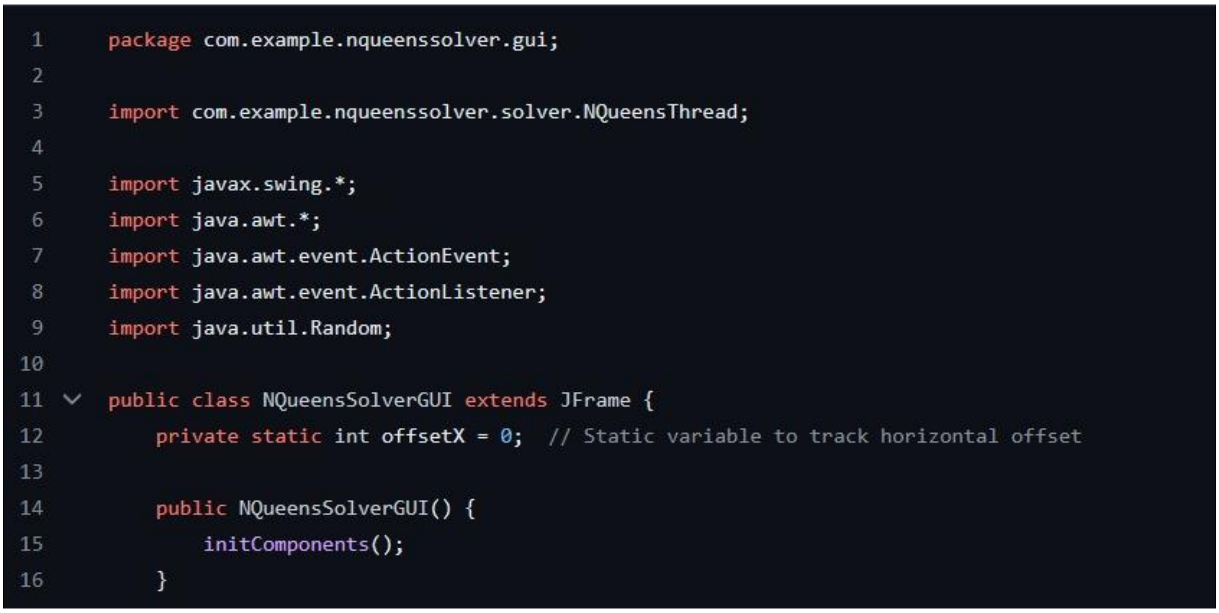
```java
    private int[] getShuffledColumns() {
        int[] columns = new int[n];
        for (int i = 0; i < n; i++) {
            columns[i] = i;
        }

        for (int i = n - 1; i > 0; i--) {
            int j = random.nextInt(i + 1);

            // Swap columns[i] and columns[j]
            int temp = columns[i];
            columns[i] = columns[j];
            columns[j] = temp;
        }

        return columns;
    }

    private boolean isSafe(int row, int col) {
        // Check if no queens are in the same column or diagonals
        for (int i = 0; i < row; i++) {
            if (solver.getQueens()[i] == col || Math.abs(solver.getQueens()[i] - col) == Math.abs(i - row)) {
                return false;
            }
        }
        return true;
    }
```



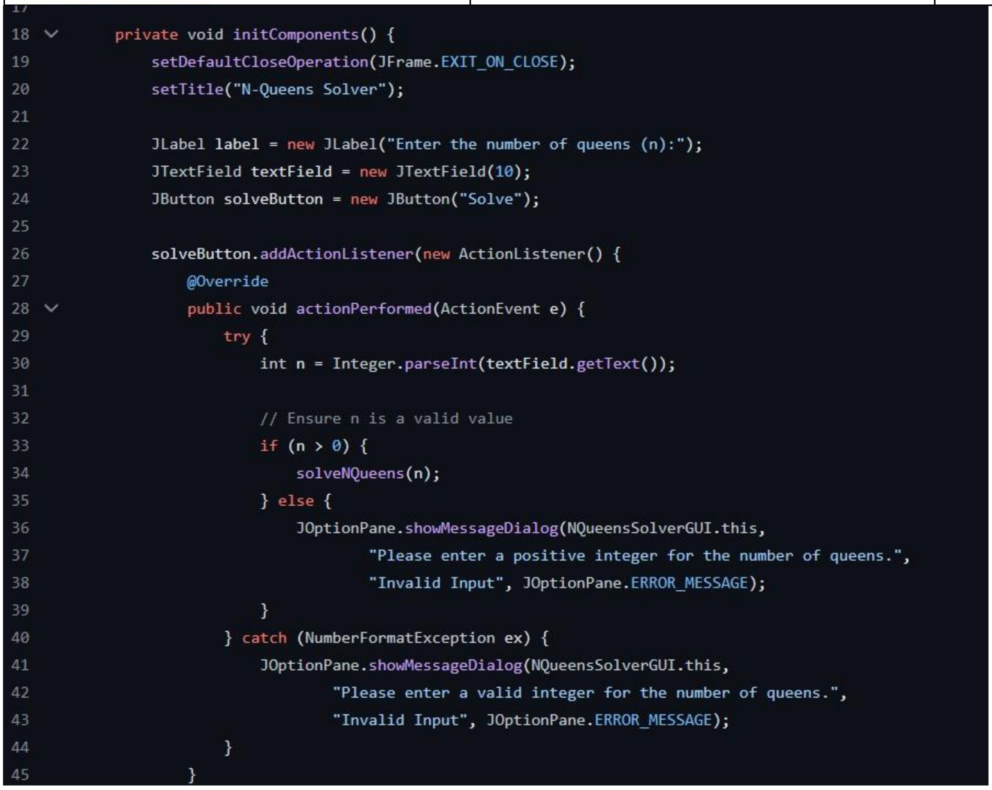
```java
        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);

            int size = queens.length;
            int cellSize = getWidth() / size;

            // Draw chessboard
            for (int i = 0; i < size; i++) {
                for (int j = 0; j < size; j++) {
                    if ((i + j) % 2 == 0) {
                        g.setColor(Color.LIGHT_GRAY);
                    } else {
                        g.setColor(Color.DARK_GRAY);
                    }
                    g.fillRect(j * cellSize, i * cellSize, cellSize, cellSize);
                }
            }

            // Draw queens
            g.setColor(Color.RED);
            for (int i = 0; i < size; i++) {
                int col = queens[i];
                g.fillOval(col * cellSize, i * cellSize, cellSize, cellSize);
            }
        }
    }
```

NQueensSolverGUI.java:

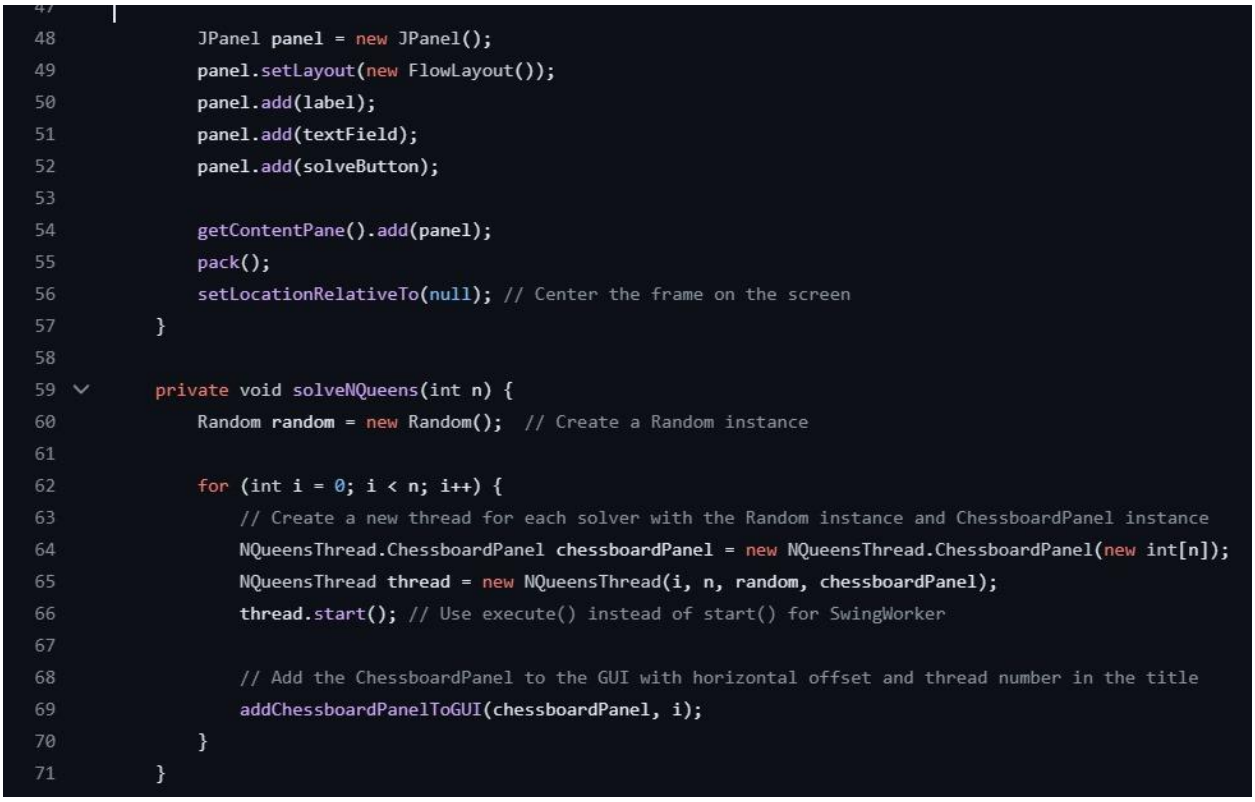
This code represents the GUI for the N-Queens solver. Users can input the number of queens, and the GUI displays solutions for each thread in separate frames with chessboard visualizations. The GUI dynamically arranges the frames to fit the screen, and each frame can be closed independently.



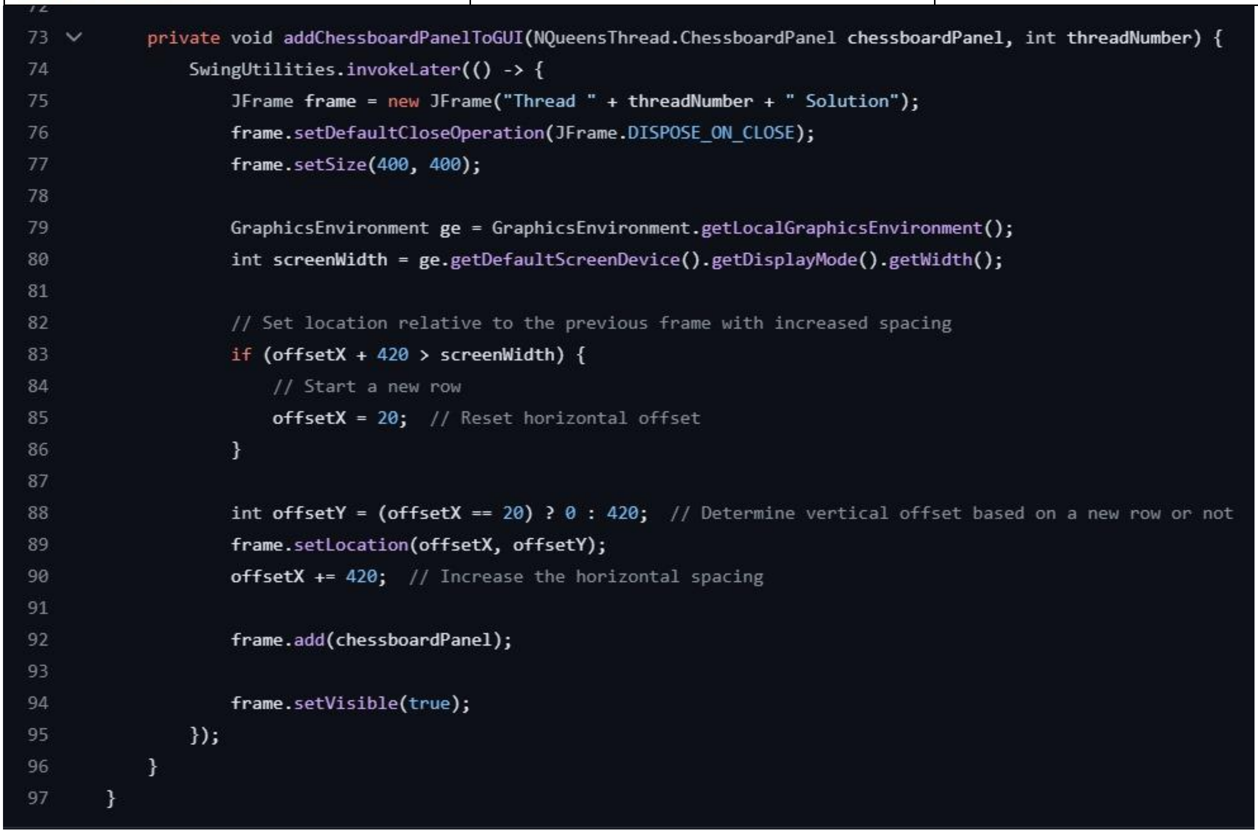
```java
package com.example.nqueenssolver.gui;

import com.example.nqueenssolver.solver.NQueensThread;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

public class NQueensSolverGUI extends JFrame {
    private static int offsetX = 0;  // Static variable to track horizontal offset

    public NQueensSolverGUI() {
        initComponents();
    }
```



```
    private void initComponents() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("N-Queens Solver");

        JLabel label = new JLabel("Enter the number of queens (n):");
        JTextField textField = new JTextField(10);
        JButton solveButton = new JButton("Solve");

        solveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    int n = Integer.parseInt(textField.getText());

                    // Ensure n is a valid value
                    if (n > 0) {
                        solveNQueens(n);
                    } else {
                        JOptionPane.showMessageDialog(NQueensSolverGUI.this,
                                "Please enter a positive integer for the number of queens.",
                                "Invalid Input", JOptionPane.ERROR_MESSAGE);
                    }
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(NQueensSolverGUI.this,
                            "Please enter a valid integer for the number of queens.",
                            "Invalid Input", JOptionPane.ERROR_MESSAGE);
                }
            }
```



```

        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        panel.add(label);
        panel.add(textField);
        panel.add(solveButton);

        getContentPane().add(panel);
        pack();
        setLocationRelativeTo(null); // Center the frame on the screen
    }

    private void solveNQueens(int n) {
        Random random = new Random();  // Create a Random instance

        for (int i = 0; i < n; i++) {
            // Create a new thread for each solver with the Random instance and ChessboardPanel instance
            NQueensThread.ChessboardPanel chessboardPanel = new NQueensThread.ChessboardPanel(new int[n]);
            NQueensThread thread = new NQueensThread(i, n, random, chessboardPanel);
            thread.start(); // Use execute() instead of start() for SwingWorker

            // Add the ChessboardPanel to the GUI with horizontal offset and thread number in the title
            addChessboardPanelToGUI(chessboardPanel, i);
        }
    }
```



```java
    private void addChessboardPanelToGUI(NQueensThread.ChessboardPanel chessboardPanel, int threadNumber) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Thread " + threadNumber + " Solution");
            frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
            frame.setSize(400, 400);

            GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
            int screenWidth = ge.getDefaultScreenDevice().getDisplayMode().getWidth();

            // Set location relative to the previous frame with increased spacing
            if (offsetX + 420 > screenWidth) {
                // Start a new row
                offsetX = 20;  // Reset horizontal offset
            }

            int offsetY = (offsetX == 20) ? 0 : 420;  // Determine vertical offset based on a new row or not
            frame.setLocation(offsetX, offsetY);
            offsetX += 420;  // Increase the horizontal spacing

            frame.add(chessboardPanel);

            frame.setVisible(true);
        });
    }
}
```