OS-2 Documentation

N Queen Problem

OVERVIEW

Welcome to the documentation for solving the N Queen Problem using Java threading! This comprehensive guide will provide you with a clear understanding of the problem statement, the significance of using threading for its solution, and how to implement it using Java. The N Queen Problem is a classic puzzle that challenges us to place N chess queens on an N×N chessboard in such a way that no two queens threaten each other. In chess, a queen can move horizontally, vertically, or diagonally across the board, making it a powerful and versatile piece. The objective is to find a configuration where no two queens can attack each other. While the N Queen Problem has intrigued minds for centuries, solving it using threading in Java adds a new dimension to its solution. Threading allows for parallel execution of tasks, enabling us to distribute the workload across multiple threads and potentially achieve significant performance improvements. By leveraging Java's threading capabilities, we can explore concurrent approaches to solving the N Queen Problem and witness the power of parallelism.

Team Members roles:

Name	ID	Role
عاصم يحيي	20210482	(GUIMain)Class, Documentation
سيف الدين محمد صالح عبد الرحمن	20210439	Project Structure,(ChessBoard)Class
على خالد	20210576	(MyTable)Class
عبدالرحمن احمد سيد محمد نصار	20210491	(NQueensSolver)Class
محمد حسن محمد حافظ	20210762	(Main, Utilities)Classes
عبدالرحمن محمد احمد صابر	20210523	(NQueensSolver)Class

Overall Approach for N Queens Problem:

The overall approach involves creating a GUI for user interaction, initiating a thread for each column to solve the N Queens Problem concurrently, and providing a visual representation of the chessboard along with a table to display the solutions found by each thread. Backtracking is used to explore the solution space, and the program handles concurrent execution and synchronization.

How Each Class Work

---

1. **GUI Initialization (GUIMain class):**
    - **A graphical user interface (GUI) is created using the Swing framework.**
    - **The user is prompted to enter the size of the chessboard (N) through a text field.**
    - **The "Submit" button triggers the initiation of the solver.**

2. **Main Class (Main class):**
    - **Creates the main frame and initializes necessary components.**
    - **Upon "Submit" button click:**
        - **Validates the input to ensure it's a positive integer.**
        - **Initializes a thread group (MyThreadGroup), a mutex, and an atomic integer to coordinate and track solutions.**
        - **Disposes of the main frame to allow the solver threads to run independently.**

        - Creates a chessboard (MyTable), and for each column in the board, creates a thread associated with a NQueensSolver instance.
        - Start each thread.

3. Table for Displaying Solutions (MyTable class):
    - Extends JFrame to create a table for displaying thread outputs.
    - Utilizes a JTable with a DefaultTableModel to dynamically add rows.
    - Provides methods to add rows to the table (addRow).

4. Utilities Class:
    - Contains utility methods for positioning frames on the screen (centerFrameOnScreen, moveFrameToLeftTop, moveFrameToRightTop, etc.).
    - Includes a delay method for thread synchronization (delay).

5. N Queens Solver (NQueensSolver class):
    - Implements the Runnable interface for concurrent execution.
    - Receives a mutex, thread group, chessboard, board size, atomic integer for solution count, and a table for display.
    - Uses backtracking to solve the N Queens Problem concurrently:
        - solveNQUtil is a recursive method exploring the solution space.
        - The main run method handles the overall flow of the solver, updates the display table, and handles interruptions.

6. Chess Board Representation (ChessBoard class):
    - Represents the chessboard visually using a GUI.

- Utilizes a 2D array to represent the logical state of the chessboard.
- Provides methods to set and empty buttons on the chessboard (setButton, emptyButton).
- Initializes the graphical representation of the chessboard with a Swing-based GUI.

Code Documentation

---

1. Main Class

```
/*
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to
edit this template
 */
```

```
package com.os.nqueenssolver;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.concurrent.Semaphore;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.locks.ReentrantLock;
import javax.swing.JOptionPane;

/**
 *
 * @author m
 */
public class Main {

    public static void main(String[] args) {
        var mainFrame = new GUIMain();
        mainFrame.setVisible(true);
        Utilities.centerFrameOnScreen(mainFrame);
        ThreadGroup tg = new ThreadGroup("MyThreadGroup");
        Object mutex = new Object();
        AtomicInteger noOfSol = new AtomicInteger(0);

mainFrame.getSubmitButton().addActionListener((ActionEvent ae)
                -> {
            String boardText =
mainFrame.getTextField().getText();
            try {
                int boardSize = Integer.parseInt(boardText);
                if (boardSize <= 0) {
                    throw new NumberFormatException();
                }
                // add code here
                Thread[] threads = new Thread[boardSize];
                mainFrame.dispose();
                MyTable table = new MyTable();
                for (int i = 0; i < boardSize; i++) {
                    ChessBoard cb = new ChessBoard(boardSize,
"Thread " + i);
                    cb.setButton(0, i);
                    threads[i] = new Thread(tg,
                            new NQueensSolver(mutex, tg, cb,
boardSize, noOfSol, table), "Thread " + i);
                    threads[i].start();
                }
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(mainFrame, "Board
size must be a valid number",
                        "Error", JOptionPane.ERROR_MESSAGE);
            }
        });

    }
}
```

2. MyTable Class

```
/*
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to
edit this template
 */
package com.os.nqueenssolver;

import java.awt.Dimension;
import java.awt.Toolkit;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
```

```
/**
 *
 * @author m
 */
public class MyTable extends JFrame{

    private final JTable table;
    private final DefaultTableModel model;
    private final JScrollPane scrollPane;

    public MyTable(){
        // Create a JFrame and a JTable
        this.table = new JTable();
        // Create a DefaultTableModel with columns and 0 rows 
initially
        this.model = new DefaultTableModel(new 
Object[]{"Successfully Terminated threads"}, 0);
        // Set the model for the table
        table.setModel(model);
        table.setDefaultEditor(Object.class, null);
        // Add the table to a JScrollPane
        scrollPane = new JScrollPane(table);
        // Add the scroll pane to the frame
        this.getContentPane().add(scrollPane);

        // Set frame properties
        this.setSize(400, 400);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Dimension screenSize = 
Toolkit.getDefaultToolkit().getScreenSize();
        int screenWidth = (int) screenSize.getWidth();
        this.setLocation(screenWidth - this.getWidth(), 0);
        this.setTitle("Threads output");
        this.setVisible(true);
    }

    public void addRow(String column1Data) {
        model.addRow(new Object[]{column1Data});
    }

}
```

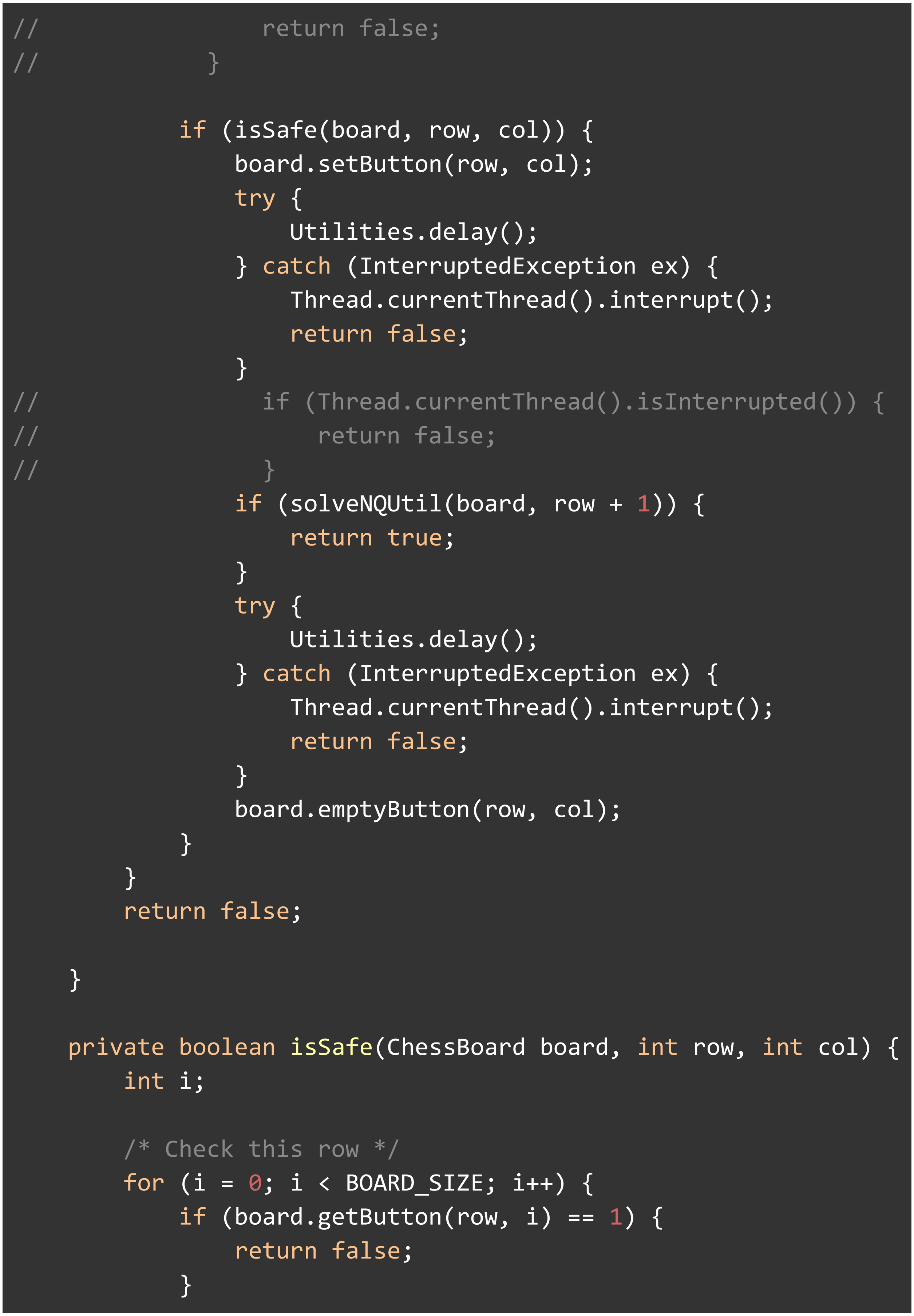
3. Utilities class

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.os.nqueenssolver;

import java.awt.Dimension;
import java.awt.Toolkit;
import javax.swing.JFrame;

/**
 *
 * @author m
 */
public class Utilities {

    public static void centerFrameOnScreen(JFrame frame) {
        // Get the screen size
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

        // Calculate the center position
        int centerX = (int) (screenSize.getWidth() - frame.getWidth()) / 2;
        int centerY = (int) (screenSize.getHeight() - frame.getHeight()) / 2;
        // Set the frame location
        frame.setLocation(centerX, centerY);
    }

    public static void moveFrameToLeftTop(JFrame frame) {
        frame.setLocation(0, 0);
    }

    public static void moveFrameToRightTop(JFrame frame) {
        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
        int rightX = (int) (screenSize.getWidth() -
frame.getWidth());
        frame.setLocation(rightX, 0);
    }

    public static void moveFrameToLeftBottom(JFrame frame) {
        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
        int bottomY = (int) (screenSize.getHeight() -
frame.getHeight());
        frame.setLocation(0, bottomY);
    }

    public static void moveFrameToRightBottom(JFrame frame) {
        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
        int rightX = (int) (screenSize.getWidth() -
frame.getWidth());
        int bottomY = (int) (screenSize.getHeight() -
frame.getHeight());
        frame.setLocation(rightX, bottomY);
    }

    public static void delay() throws InterruptedException {

        Thread.sleep(600);

    }
}
```

4. NQueensSolver class

```
/*
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to
edit this template
 */
package com.os.nqueenssolver;

import java.util.concurrent.atomic.AtomicInteger;
import javax.swing.JOptionPane;

/**
 *
 * @author m
 */
public class NQueensSolver implements Runnable {

    private final int BOARD_SIZE;
    private final ChessBoard cb;
    private final ThreadGroup group;
    private final Object mutex;
    private volatile AtomicInteger noOfSol;
    private final MyTable table;

    public NQueensSolver(Object mutex, ThreadGroup group,
ChessBoard cb, int boardSize, AtomicInteger noOfSol, MyTable
table) {
        BOARD_SIZE = boardSize;
        this.cb = cb;
        this.group = group;
        this.mutex = mutex;
        this.noOfSol = noOfSol;
        this.table = table;
    }

    @Override
    public void run() {
//        if (BOARD_SIZE == 1 || BOARD_SIZE == 2 || BOARD_SIZE
== 3) {
//
//            try {
//                mutex.acquire();
//            } catch (InterruptedException ex) {
//                return;
//            }
//            System.out.println("NO Solution can be found");
//             JOptionPane.showMessageDialog(cb, "NO Solution
can be found",
//                    "Error", JOptionPane.INFORMATION_MESSAGE);
//            group.interrupt();
//            mutex.release();
//            return;
//        }

        if (solveNQUtil(cb, 1)) {

            if (noOfSol.compareAndSet(0, 1)) {
                table.addRow(Thread.currentThread().getName() +
" is the first"
                        + " thread to find a solution!");
                JOptionPane.showMessageDialog(cb,
Thread.currentThread().getName() + " is the first"
                        + " thread to find a solution!",
                        "Complete",
JOptionPane.INFORMATION_MESSAGE);
```

```
            } else {
                table.addRow(Thread.currentThread().getName() +
" found solution number " + noOfSol.incrementAndGet());
            }

        } else if (Thread.currentThread().isInterrupted()) {
            table.addRow(Thread.currentThread().getName() + "
was interrupted");
        } else {
            table.addRow(Thread.currentThread().getName()
                    + " Couldn't Find a solution");
        }
    }

    boolean solveNQUtil(ChessBoard board, int row) {

//        if (Thread.currentThread().isInterrupted()) {
//            return false;
//        }
        if (row == BOARD_SIZE) {
//            if (Thread.currentThread().isInterrupted()) {
//                return false;
//            }

//            synchronized(mutex){
//                if (Thread.currentThread().isInterrupted()) {
//                    return false;
//                }
//                group.interrupt();
//
System.out.println(Thread.currentThread().getName() + " found a
solution");
//            }
            return true;
        }

        for (int col = 0; col < BOARD_SIZE; col++) {
//            if (Thread.currentThread().isInterrupted()) {
```

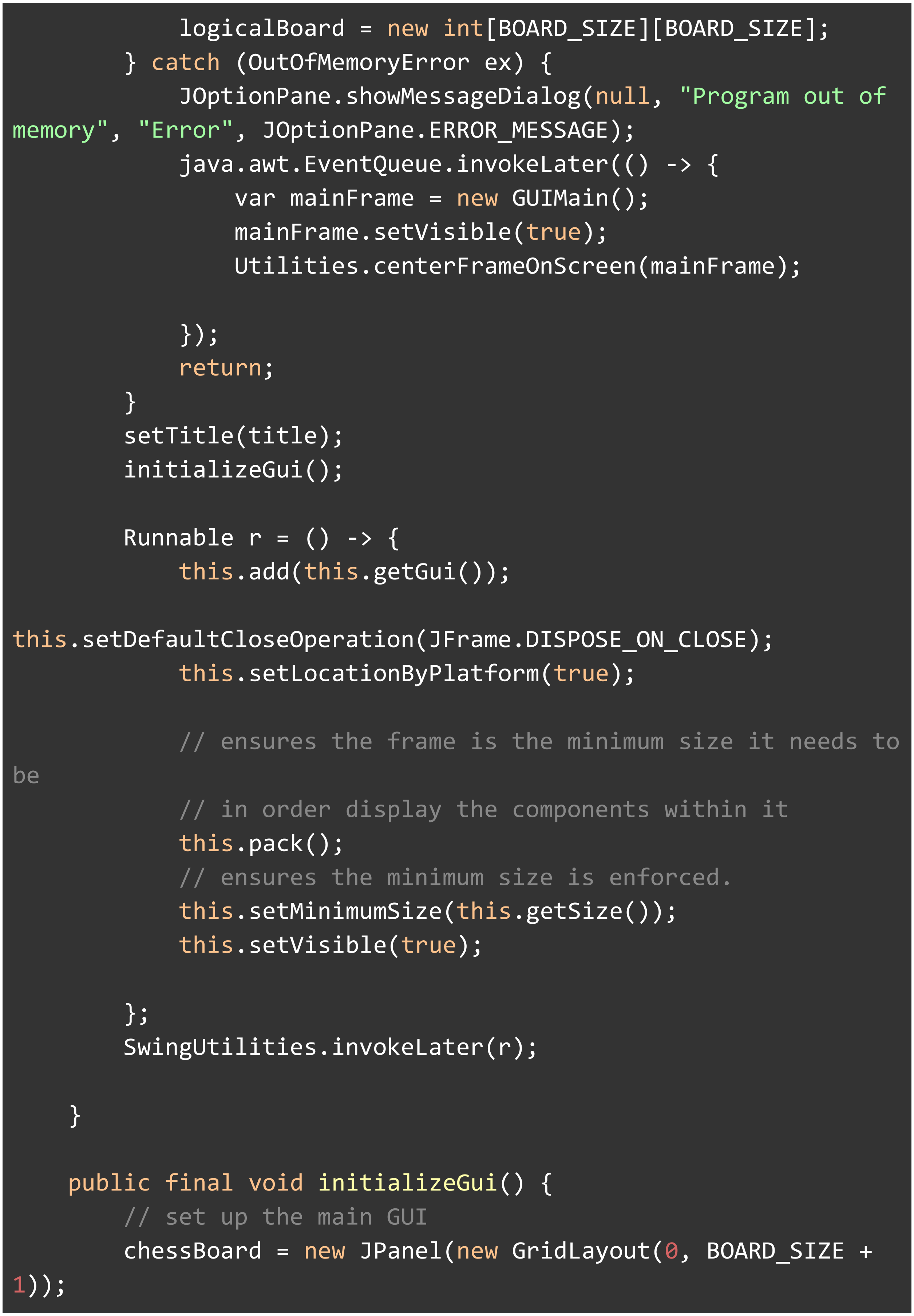


```
//                return false;
//            }

            if (isSafe(board, row, col)) {
                board.setButton(row, col);
                try {
                    Utilities.delay();
                } catch (InterruptedException ex) {
                    Thread.currentThread().interrupt();
                    return false;
                }
//                if (Thread.currentThread().isInterrupted()) {
//                    return false;
//                }
                if (solveNQUtil(board, row + 1)) {
                    return true;
                }
                try {
                    Utilities.delay();
                } catch (InterruptedException ex) {
                    Thread.currentThread().interrupt();
                    return false;
                }
                board.emptyButton(row, col);
            }
        }
        return false;

    }

    private boolean isSafe(ChessBoard board, int row, int col) {
        int i;

        /* Check this row */
        for (i = 0; i < BOARD_SIZE; i++) {
            if (board.getButton(row, i) == 1) {
                return false;
            }
    }

    /*check column*/
    for (i = 0; i < BOARD_SIZE; i++) {
      if (board.getButton(i, col) == 1) {
        return false;
      }
    }

    for (i = 0; i < BOARD_SIZE; i++) {
      int diagonalRow = row - col + i;
      int diagonalCol = i;

      if (diagonalRow >= 0 && diagonalRow < BOARD_SIZE && diagonalCol >= 0 && diagonalCol < BOARD_SIZE) {
        if (cb.getButton(diagonalRow, diagonalCol) == 1) {
          return false;
        }
      }
    }

    // Check secondary diagonal
    for (i = 0; i < BOARD_SIZE; i++) {
      int diagonalRow = row + col - i;
      int diagonalCol = i;

      if (diagonalRow >= 0 && diagonalRow < BOARD_SIZE && diagonalCol >= 0 && diagonalCol < BOARD_SIZE) {
        if (cb.getButton(diagonalRow, diagonalCol) == 1) {
          return false;
        }
      }
    }

    return true;
  }
```

```
}
```

5. ChessBoard class

```
package com.os.nqueenssolver;

import java.awt.*;
import java.awt.image.BufferedImage;

import javax.swing.*;
import javax.swing.border.*;

public class ChessBoard extends javax.swing.JFrame {

    private final int BOARD_SIZE;
    private final JPanel gui = new JPanel(new BorderLayout(3,
3));
    private JButton[][] chessBoardSquares;
    private int[][] logicalBoard;
    private JPanel chessBoard;
    private final String userDir =
System.getProperty("user.dir");
    private final ImageIcon icon = new ImageIcon(
            new BufferedImage(64, 64,
BufferedImage.TYPE_INT_ARGB));
    private final ImageIcon img = new ImageIcon(userDir +
"/images/queen.png");

    ChessBoard(int boardSize, String title) {

        BOARD_SIZE = boardSize;
        try {
            chessBoardSquares = new
JButton[BOARD_SIZE][BOARD_SIZE];
```



```
            logicalBoard = new int[BOARD_SIZE][BOARD_SIZE];
        } catch (OutOfMemoryError ex) {
            JOptionPane.showMessageDialog(null, "Program out of
memory", "Error", JOptionPane.ERROR_MESSAGE);
            java.awt.EventQueue.invokeLater(() -> {
                var mainFrame = new GUIMain();
                mainFrame.setVisible(true);
                Utilities.centerFrameOnScreen(mainFrame);

            });
            return;
        }
        setTitle(title);
        initializeGui();

        Runnable r = () -> {
            this.add(this.getGui());

this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
            this.setLocationByPlatform(true);

            // ensures the frame is the minimum size it needs to
be
            // in order display the components within it
            this.pack();
            // ensures the minimum size is enforced.
            this.setMinimumSize(this.getSize());
            this.setVisible(true);

        };
        SwingUtilities.invokeLater(r);

    }

    public final void initializeGui() {
        // set up the main GUI
        chessBoard = new JPanel(new GridLayout(0, BOARD_SIZE +
1));
```

```
        chessBoard.setBorder(new LineBorder(Color.BLACK));
        gui.add(chessBoard);
        // create the chess board squares
        Insets buttonMargin = new Insets(0, 0, 0, 0);
        for (int i = 0; i < chessBoardSquares.length; i++) {
            for (int j = 0; j < chessBoardSquares[i].length;
j++) {
                JButton b = new JButton();
                b.setMargin(buttonMargin);
                // our chess pieces are 64x64 px in size, so
we'll
                // 'fill this in' using a transparent icon..
                b.setIcon(icon);
                if ((j % 2 != 0 && i % 2 != 0) || (j % 2 == 0 &&
i % 2 == 0)) {
                    b.setBackground(Color.WHITE);
                } else {
                    b.setBackground(Color.DARK_GRAY);
                }
                chessBoardSquares[i][j] = b;
            }
        }

        //fill the chess board
        // fill the black non-pawn piece row
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                switch (j) {
                    case 0:
                        chessBoard.add(new JLabel(""));
                    default:
                        chessBoard.add(chessBoardSquares[i][j]);
                }
            }
        }
    }

    public final JComponent getChessBoard() {
        return chessBoard;
    }

    public void setButton(int row, int col) {
        chessBoardSquares[row][col].setIcon(img);
        logicalBoard[row][col] = 1;
    }

    public void emptyButton(int row, int col) {
        chessBoardSquares[row][col].setIcon(icon);
        logicalBoard[row][col] = 0;
    }

    public int getButton(int row, int col) {
        return logicalBoard[row][col];
    }

    public final JComponent getGui() {
        return gui;
    }

}
```

6. GUIMain class

```
/*
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
 * Click
nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to
edit this template
```

```java
 */
package com.os.nqueenssolver;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JTextField;

/**
 *
 * @author m
 */
public class GUIMain extends javax.swing.JFrame {

    /**
     * Creates new form NewJFrame
     */
    public GUIMain() {
        initComponents();
        textField.addActionListener((ActionEvent e) -> {
            // Perform the action when Enter is pressed
            submit.doClick();
        });
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        submit = new javax.swing.JButton();
        textField = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLO
SE);
        setTitle("N Queens Solver");

        jPanel1.setBackground(new java.awt.Color(204, 204,
204));
        jPanel1.setBorder(new
javax.swing.border.MatteBorder(null));

        jLabel1.setFont(new java.awt.Font("Segoe UI", 1, 18));
// NOI18N
        jLabel1.setForeground(new java.awt.Color(0, 0, 0));

jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER
);
        jLabel1.setText("Welcome to N Queens Solver");

        submit.setText("Submit");
        submit.addActionListener(new
java.awt.event.ActionListener() {
            public void
actionPerformed(java.awt.event.ActionEvent evt) {
                submitActionPerformed(evt);
            }
        });

        textField.setBackground(new java.awt.Color(255, 255,
255));
        textField.setForeground(new java.awt.Color(102, 102,
102));
        textField.setToolTipText("Enter a valid number");
        textField.addActionListener(new
java.awt.event.ActionListener() {
            public void
actionPerformed(java.awt.event.ActionEvent evt) {
                textFieldActionPerformed(evt);
            }
        });

        jLabel2.setFont(new java.awt.Font("Segoe UI", 0, 14));
// NOI18N
        jLabel2.setForeground(new java.awt.Color(0, 0, 0));
        jLabel2.setText("Enter the size of the board: ");

        javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(120, 120, 120)
                .addComponent(jLabel1,
javax.swing.GroupLayout.PREFERRED_SIZE, 383,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
                .addGap(0, 68, Short.MAX_VALUE)
                .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELA
TED)
                .addComponent(textField,
javax.swing.GroupLayout.PREFERRED_SIZE, 148,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                .addGap(235, 235, 235))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addComponent(submit)
                .addGap(274, 274, 274))
        );
        jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(24, 24, 24)
                .addComponent(jLabel1,
javax.swing.GroupLayout.PREFERRED_SIZE, 37,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELA
TED, 54, Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLay
out.Alignment.BASELINE)
                    .addComponent(textField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel2))
                .addGap(18, 18, 18)
                .addComponent(submit)
                .addGap(102, 102, 102))
        );

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
DING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(0, 0, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
DING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void
textFieldActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_textFieldActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_textFieldActionPerformed

    private void
submitActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_submitActionPerformed
        // TODO add your handling code here:

    }//GEN-LAST:event_submitActionPerformed
    public JButton getSubmitButton() {
        return submit;
    }

    public JTextField getTextField() {
        return textField;
    }

    // Variables declaration - do not modify//GEN-
BEGIN:variables
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JButton submit;
    private javax.swing.JTextField textField;
    // End of variables declaration//GEN-END:variables
}
```