```java
/**
     * The text fields representing the cells of the Sudoku grid.
     */
    private TextField[][] tfCells = new TextField[9][9];

```

```java
/**
     * Starts the Sudoku game by initializing the GUI components and displaying the game window.
     *
     * @param primaryStage the primary stage for the application
     */
    @Override
    public void start(Stage primaryStage) {
        GridPane grid = new GridPane();

        // Create and configure the text fields for the Sudoku grid
        for (int row = 0; row < 9; row++) {
            for (int col = 0; col < 9; col++) {
                tfCells[row][col] = new TextField();
                tfCells[row][col].setPrefSize(50, 50);
                tfCells[row][col].setText(PuzzleToSolve[row][col] == 0 ? "" : String.valueOf(PuzzleToSolve[row][col]));
                tfCells[row][col].setStyle("-fx-font-size: 20;");
                grid.add(tfCells[row][col], col, row);
            }
        }

        // Create and configure the solve button
        Button solveButton = new Button("Solve");
        solveButton.setPrefSize(450, 50);
        solveButton.setOnAction(e -> solveSudoku());

        // Create the scene and add the grid and solve button to it
        Scene scene = new Scene(grid, WINDOW_WIDTH, WINDOW_HEIGHT);
        grid.add(solveButton, 0, 9, 9, 1);

        // Configure the primary stage and show it
        primaryStage.setTitle("Sudoku Game");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

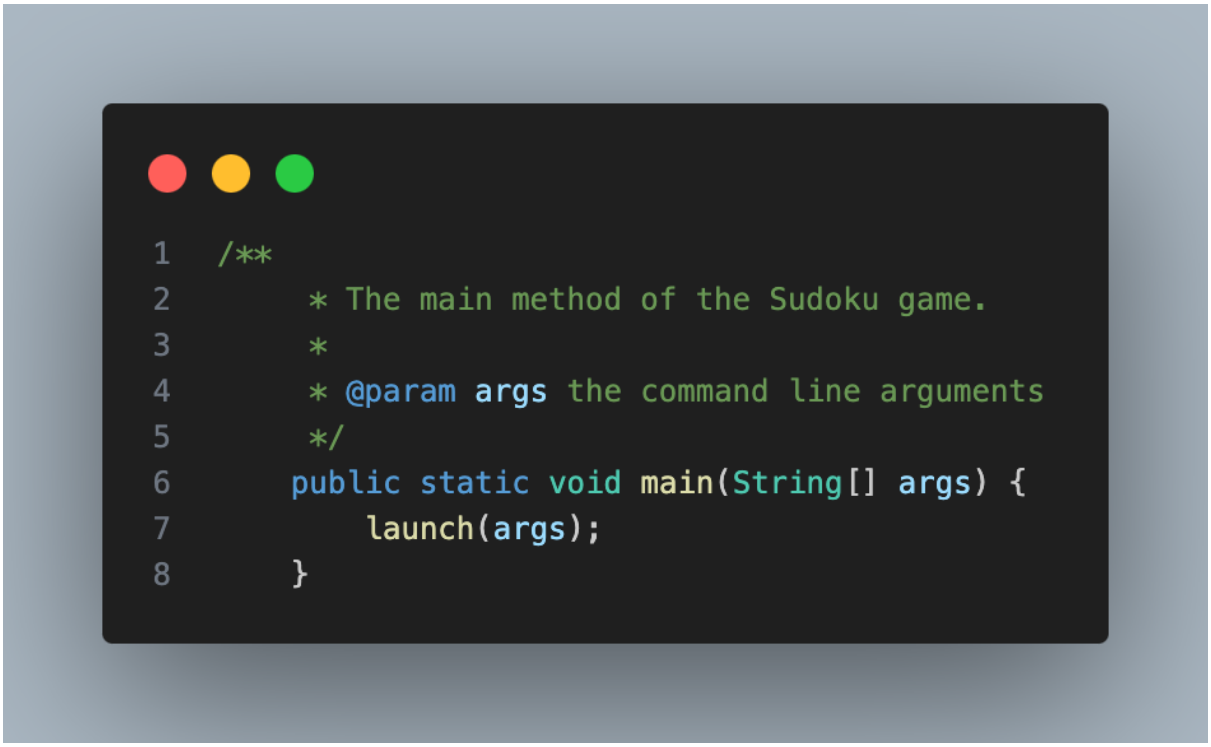
```java
/**
     * Solves the Sudoku puzzle by backtracking.
     * Updates the text fields with the solved values if a solution is found.
     * Displays an error message if the puzzle is invalid.
     */
    private void solveSudoku() {
        int[][] board = new int[9][9];

        // Get the values from the text fields and populate the board
        for (int row = 0; row < 9; row++) {
            for (int col = 0; col < 9; col++) {
                String value = tfCells[row][col].getText();
                board[row][col] = value.isEmpty() ? 0 : Integer.parseInt(value);
            }
        }

        // Solve the puzzle using backtracking
        if (solve(board)) {
            // Update the text fields with the solved values
            for (int row = 0; row < 9; row++) {
                for (int col = 0; col < 9; col++) {
                    tfCells[row][col].setText(String.valueOf(board[row][col]));
                }
            }
        } else {
            // Display an error message if the puzzle is invalid
            showAlert("Invalid Sudoku", "The given Sudoku puzzle is invalid.");
        }
    }
```

```java
/**
     * Solves the Sudoku puzzle using backtracking.
     *
     * @param board the Sudoku puzzle grid
     * @return true if a solution is found, false otherwise
     */
    private boolean solve(int[][] board) {
        for (int row = 0; row < 9; row++) {
            for (int col = 0; col < 9; col++) {
                if (board[row][col] == 0) {
                    for (int num = 1; num <= 9; num++) {
                        if (isValid(board, row, col, num)) {
                            board[row][col] = num;
                            if (solve(board)) {
                                return true;
                            }
                            board[row][col] = 0; // backtrack
                        }
                    }
                    return false; // no valid number found
                }
            }
        }
        return true; // all cells are filled
    }
```

```java
/**
     * Checks if a number is valid in the Sudoku puzzle grid.
     *
     * @param board the Sudoku puzzle grid
     * @param row   the row index of the cell
     * @param col   the column index of the cell
     * @param num   the number to be checked
     * @return true if the number is valid, false otherwise
     */
    private boolean isValid(int[][] board, int row, int col, int num) {
        // Check if the number is not in the current row and column
        for (int i = 0; i < 9; i++) {
            if (board[row][i] == num || board[i][col] == num) {
                return false;
            }
        }

        // Check if the number is not in the current 3x3 grid
        int startRow = row - row % 3;
        int startCol = col - col % 3;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i + startRow][j + startCol] == num) {
                    return false;
                }
            }
        }

        return true;
    }
```

```java
/**
     * Displays an alert dialog with the specified title and message.
     *
     * @param title   the title of the alert dialog
     * @param message the message of the alert dialog
     */
    private void showAlert(String title, String message) {
        Alert alert = new Alert(AlertType.ERROR);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }

```

```java
/**
     * The root pane of the Sudoku game.
     */
    public GridPane root;

    /**
     * Returns the root pane of the Sudoku game.
     * If the root pane is null, starts the game and initializes the root pane.
     *
     * @return the root pane of the Sudoku game
     */
    public GridPane getRoot() {
        if (root == null) {
            start(new Stage());
            root = new GridPane();
        }
        return root;
    }
```



```java
/**
     * The main method of the Sudoku game.
     *
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }
```