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

public class TicTacToe extends JFrame {
    private JButton[][] buttons = new JButton[3][3];
    private String playerX;
    private String playerO;
    private String currentPlayer;
    private String symbolX = "X";
    private String symbolO = "O";
    private JLabel scoreLabel;
    private int movesCount;
    private int scoreX;
    private int scoreO;
    private boolean singlePlayerMode;

    public TicTacToe() {
        setTitle("Tic Tac Toe");
        setSize(400, 500);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        createMenu();
```

```
        initGame();
        createButtons();

        scoreLabel = new JLabel("Score - " + playerX + ": " + scoreX + " | " + playerO + ": " +
scoreO, SwingConstants.CENTER);
        add(scoreLabel, BorderLayout.SOUTH);

        setVisible(true);
    }

    private void createMenu() {
        JMenuBar menuBar = new JMenuBar();
        JMenu gameMenu = new JMenu("Game");
        JMenuItem newGameItem = new JMenuItem("New Game");
        JMenuItem resetScoresItem = new JMenuItem("Reset Scores");
        JMenuItem exitItem = new JMenuItem("Exit");

        newGameItem.addActionListener(e -> resetGame());
        resetScoresItem.addActionListener(e -> resetScores());
        exitItem.addActionListener(e -> System.exit(0));

        gameMenu.add(newGameItem);
        gameMenu.add(resetScoresItem);
        gameMenu.add(exitItem);
        menuBar.add(gameMenu);
        setJMenuBar(menuBar);
```

```
    }

    private void initGame() {
        playerX = JOptionPane.showInputDialog(this, "Enter name for Player X:");
        String mode = JOptionPane.showInputDialog(this, "Choose mode: Type '1' for Player vs Player or '2' for Player vs Computer:");
        singlePlayerMode = mode.equals("2");
        playerO = singlePlayerMode ? "Computer" : JOptionPane.showInputDialog(this, "Enter name for Player O:");
        chooseSymbols();
        currentPlayer = symbolX;
        movesCount = 0;
        scoreX = 0;
        scoreO = 0;
    }

    private void chooseSymbols() {
        Object[] options = {"X", "O", "Custom"};
        int choice = JOptionPane.showOptionDialog(this, "Choose your symbols:", "Symbol Selection",
                JOptionPane.DEFAULT_OPTION, JOptionPane.INFORMATION_MESSAGE, null, options, options[0]);

        if (choice == 2) {
            symbolX = JOptionPane.showInputDialog(this, "Enter symbol for Player X (1 character):");
            while (symbolX.length() != 1) {
```

```
            symbolX = JOptionPane.showInputDialog(this, "Please enter a single character for Player X:");
        }
        symbolO = JOptionPane.showInputDialog(this, "Enter symbol for Player O (1 character):");
        while (symbolO.length() != 1 || symbolO.equals(symbolX)) {
            symbolO = JOptionPane.showInputDialog(this, "Please enter a different single character for Player O:");
        }
    } else {
        symbolX = choice == 0 ? "X" : "O";
        symbolO = choice == 0 ? "O" : "X";
    }
  }

  private void createButtons() {
    JPanel buttonPanel = new JPanel();
    buttonPanel.setLayout(new GridLayout(3, 3));
    for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
        buttons[i][j] = new JButton();
        buttons[i][j].setFont(new Font("Arial", Font.PLAIN, 60));
        buttons[i][j].setBackground(Color.LIGHT_GRAY);
        buttons[i][j].addActionListener(new ButtonClickListener(i, j));
        buttonPanel.add(buttons[i][j]);
      }
    }
```

```
        add(buttonPanel, BorderLayout.CENTER);
    }

    private class ButtonClickListener implements ActionListener {
        private final int row, col;

        public ButtonClickListener(int row, int col) {
            this.row = row;
            this.col = col;
        }

        @Override
        public void actionPerformed(ActionEvent e) {
            if (buttons[row][col].getText().equals("") && movesCount < 9) {
                buttons[row][col].setText(currentPlayer);
                movesCount++;
                if (checkForWinner()) {
                    updateScore();
                    JOptionPane.showMessageDialog(null, (currentPlayer.equals(symbolX) ? playerX
: playerO) + " wins!");
                    resetGame();
                } else if (movesCount == 9) {
                    JOptionPane.showMessageDialog(null, "It's a draw!");
                    resetGame();
                } else {
                    currentPlayer = currentPlayer.equals(symbolX) ? symbolO : symbolX;
```

```
            if (singlePlayerMode && currentPlayer.equals(symbolO)) {
                computerMove();  // The computer takes its turn in single-player mode
            }
        }
    }
}

private boolean checkForWinner() {
    // Check rows, columns, and diagonals
    for (int i = 0; i < 3; i++) {
        if (buttons[i][0].getText().equals(currentPlayer) &&
            buttons[i][1].getText().equals(currentPlayer) &&
            buttons[i][2].getText().equals(currentPlayer)) {
            return true;
        }
        if (buttons[0][i].getText().equals(currentPlayer) &&
            buttons[1][i].getText().equals(currentPlayer) &&
            buttons[2][i].getText().equals(currentPlayer)) {
            return true;
        }
    }
    if (buttons[0][0].getText().equals(currentPlayer) &&
        buttons[1][1].getText().equals(currentPlayer) &&
        buttons[2][2].getText().equals(currentPlayer)) {
        return true;
```

```
    }
    return buttons[0][2].getText().equals(currentPlayer) &&
            buttons[1][1].getText().equals(currentPlayer) &&
            buttons[2][0].getText().equals(currentPlayer);
}

private void computerMove() {
    Random rand = new Random();
    int row, col;
    do {
        row = rand.nextInt(3);
        col = rand.nextInt(3);
    } while (!buttons[row][col].getText().equals(""));

    buttons[row][col].setText(currentPlayer);
    movesCount++;
    if (checkForWinner()) {
        updateScore();
        JOptionPane.showMessageDialog(null, playerO + " wins!");
        resetGame();
    } else if (movesCount == 9) {
        JOptionPane.showMessageDialog(null, "It's a draw!");
        resetGame();
    } else {
        currentPlayer = symbolX; // Switch back to player X
    }
```

```
}

private void updateScore() {
    if (currentPlayer.equals(symbolX)) {
        scoreX++;
    } else {
        scoreO++;
    }
    scoreLabel.setText("Score - " + playerX + ": " + scoreX + " | " + playerO + ": " + scoreO);
}

private void resetGame() {
    movesCount = 0;
    currentPlayer = symbolX;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            buttons[i][j].setText("");
        }
    }
}

private void resetScores() {
    scoreX = 0;
    scoreO = 0;
    scoreLabel.setText("Score - " + playerX + ": " + scoreX + " | " + playerO + ": " + scoreO);
}
```

```
    public static void main(String[] args) {
        SwingUtilities.invokeLater(TicTacToe::new);
    }
}
```