import javax.swing.\*;  
import javax.swing.border.Border;  
import java.awt.\*;  
import java.awt.event.KeyEvent;  
import java.awt.event.KeyListener;  
import java.util.Random;  
public class Main2048 extends JFrame implements KeyListener {  
    private static final int BOARD\_SIZE = 5;  
    private static final int TILE\_SIZE = 100;  
    private static int[][] board = new int[BOARD\_SIZE][BOARD\_SIZE];  
    private static Random rand = new Random();  
    private static boolean isGameOver = false;  
    private static int score = 0;  
     private JPanel gamePanel;  
    private JLabel[][] tileLabels;  
   public Main2048() {  
        setTitle("2048 Game");  
        setSize(BOARD\_SIZE \* TILE\_SIZE, BOARD\_SIZE \* TILE\_SIZE);  
        setLocationRelativeTo(null);  
        setDefaultCloseOperation(EXIT\_ON\_CLOSE);  
        setResizable(false);  
        addKeyListener(this);  
        initializeBoard();  
        createUI();  
    }  
    private void initializeBoard() {  
        placeRandomTile();  
        placeRandomTile();  
    }  
    private void placeRandomTile() {  
        int emptyCells = 0;  
        for (int[] row : board) {  
            for (int cell : row) {  
                if (cell == 0) {  
                    emptyCells++;  
                }  
            }  
        }  
        if (emptyCells == 0) return;  
        int randomIndex = rand.nextInt(emptyCells);  
        int value = rand.nextInt(10) == 0 ? 4 : 2;  
  
        emptyCells = 0;  
        for (int i = 0; i < BOARD\_SIZE; i++) {  
            for (int j = 0; j < BOARD\_SIZE; j++) {  
                if (board[i][j] == 0) {  
                    if (emptyCells == randomIndex) {  
                        board[i][j] = value;  
                        return;  
                    }  
                    emptyCells++;  
                }  
            }  
        }  
    }  
    private void createUI() {  
        gamePanel = new JPanel(new GridLayout(BOARD\_SIZE, BOARD\_SIZE));  
        tileLabels = new JLabel[BOARD\_SIZE][BOARD\_SIZE];  
        for (int i = 0; i < BOARD\_SIZE; i++) {  
            for (int j = 0; j < BOARD\_SIZE; j++) {  
                tileLabels[i][j] = new JLabel();  
                tileLabels[i][j].setPreferredSize(new Dimension(TILE\_SIZE, TILE\_SIZE));  
                tileLabels[i][j].setHorizontalAlignment(SwingConstants.CENTER);  
                tileLabels[i][j].setFont(new Font("Arial", Font.BOLD, 24));  
                updateTileLabel(i, j);  
                gamePanel.add(tileLabels[i][j]);  
            }  
        }  
        addGridLines();  
        updateScore();  
        add(gamePanel, BorderLayout.CENTER);  
    }  
    private void addGridLines() {  
        Border lightBrownBorder = BorderFactory.createLineBorder(new Color(191, 144, 0), 4);  
        Border emptyBorder = BorderFactory.createEmptyBorder(3, 3, 3, 3);  
        Border doubleBorder = BorderFactory.createCompoundBorder(lightBrownBorder, emptyBorder);  
  
        for (int i = 0; i < BOARD\_SIZE; i++) {  
            for (int j = 0; j < BOARD\_SIZE; j++) {  
                if (i != 0 && j != 0) {  
                    tileLabels[i][j].setBorder(doubleBorder);  
                } else {  
                    tileLabels[i][j].setBorder(lightBrownBorder);  
                }  
            }  
        }  
    }  
    private void updateTileLabel(int row, int col) {  
        int value = board[row][col];  
        String text = value > 0 ? String.valueOf(value) : "";  
        tileLabels[row][col].setText(text);  
        tileLabels[row][col].setBackground(getTileColor(value));  
        tileLabels[row][col].setOpaque(true);  
    }  
    private Color getTileColor(int value) {  
        switch (value) {  
            case 2:  
                return new Color(238, 228, 218);  
            case 4:  
                return new Color(237, 224, 200);  
            case 8:  
                return new Color(242, 177, 121);  
            case 16:  
                return new Color(245, 149, 99);  
            case 32:  
                return new Color(246, 124, 95);  
            case 64:  
                return new Color(246, 94, 59);  
            case 128:  
                return new Color(237, 207, 114);  
            case 256:  
                return new Color(237, 204, 97);  
            case 512:  
                return new Color(237, 200, 80);  
            case 1024:  
                return new Color(237, 197, 63);  
            case 2048:  
                return new Color(237, 194, 46);  
            default:  
                return new Color(205, 193, 180);  
        }  
    }  
    private void updateScore() {  
        JLabel scoreLabel = new JLabel("Score: " + score);  
        scoreLabel.setFont(new Font("Arial", Font.PLAIN, 18));  
        Component[] components = getContentPane().getComponents();//remove  
        for (Component component : components) {  
            if (component instanceof JLabel) {  
                String labelText = ((JLabel) component).getText();  
                if (labelText.startsWith("Score: ")) {  
                    remove(component);  
                }  
            }  
        }      
        add(scoreLabel, BorderLayout.NORTH);  
        validate();  
    }  
  
    private void updateBoardUI() {  
        for (int i = 0; i < BOARD\_SIZE; i++) {  
            for (int j = 0; j < BOARD\_SIZE; j++) {  
                updateTileLabel(i, j);  
            }  
        }  
        gamePanel.revalidate();  
        gamePanel.repaint();  
    }  
    @Override  
    public void keyPressed(KeyEvent e) {  
        if (!isGameOver) {  
            int keyCode = e.getKeyCode();  
            boolean validMove = false;  
            switch (keyCode) {  
                case KeyEvent.VK\_UP:  
                    validMove = moveTiles("W");  
                    break;  
                case KeyEvent.VK\_LEFT:  
                    validMove = moveTiles("A");  
                    break;  
                case KeyEvent.VK\_DOWN:  
                    validMove = moveTiles("S");  
                    break;  
                case KeyEvent.VK\_RIGHT:  
                    validMove = moveTiles("D");  
                    break;  
            }  
  
            if (validMove) {  
                placeRandomTile();  
                updateScore();  
                updateBoardUI();  
                isGameOver = isGameOver();  
                if (isGameOver) {  
                    JOptionPane.showMessageDialog(this, "Game Over!");  
                }  
                if (hasReached2048()) {  
                    JOptionPane.showMessageDialog(this, "Player Won!");  
                    isGameOver = true;  
                }  
            }  
        }  
    }  
    private boolean moveTiles(String direction) {  
        int[][] newBoard = new int[BOARD\_SIZE][BOARD\_SIZE];  
        for (int i = 0; i < BOARD\_SIZE; i++) {  
            System.arraycopy(board[i], 0, newBoard[i], 0, BOARD\_SIZE);  
        }  
  
        boolean validMove = moveTiles(newBoard, direction);  
  
        if (validMove) {  
            board = newBoard;  
            return true;  
        }  
        return false;  
    }  
    private boolean moveTiles(int[][] board, String direction) {  
        switch (direction) {  
            case "W":  
                for (int j = 0; j < BOARD\_SIZE; j++) {  
                    int[] column = getColumn(j);  
                    column = mergeTiles(column);  
                    setColumn(board, j, column);  
                }  
                break;  
            case "A":  
                for (int i = 0; i < BOARD\_SIZE; i++) {  
                    int[] row = board[i];  
                    row = mergeTiles(row);  
                    board[i] = row;  
                }  
                break;  
            case "S":  
                for (int j = 0; j < BOARD\_SIZE; j++) {  
                    int[] column = getColumn(j);  
                    column = reverseArray(mergeTiles(reverseArray(column)));  
                    setColumn(board, j, column);  
                }  
                break;  
            case "D":  
                for (int i = 0; i < BOARD\_SIZE; i++) {  
                    int[] row = board[i];  
                    row = reverseArray(mergeTiles(reverseArray(row)));  
                    board[i] = row;  
                }  
                break;  
            default:  
                throw new IllegalArgumentException("Invalid direction: " + direction);  
        }  
        return !isSameBoard(this.board, board);  
    }  
    private int[] getColumn(int index) {  
        int[] column = new int[BOARD\_SIZE];  
        for (int i = 0; i < BOARD\_SIZE; i++) {  
            column[i] = this.board[i][index];  
        }  
        return column;  
    }  
    private void setColumn(int[][] board, int index, int[] column) {  
        for (int i = 0; i < BOARD\_SIZE; i++) {  
            board[i][index] = column[i];  
        }  
    }  
    private int[] mergeTiles(int[] row) {  
        int[] mergedRow = new int[BOARD\_SIZE];  
        int mergeIndex = 0;  
        for (int i = 0; i < BOARD\_SIZE; i++) {  
            if (row[i] != 0) {  
                if (i < BOARD\_SIZE - 1 && row[i] == row[i + 1]) {  
                    mergedRow[mergeIndex++] = row[i] \* 2;  
                    score += row[i] \* 2;  
                    i++; // Skip  
                } else {  
                    mergedRow[mergeIndex++] = row[i];  
                }  
            }  
        }  
        return mergedRow;  
    }  
    private int[] reverseArray(int[] array) {  
        int[] reversedArray = new int[array.length];  
        for (int i = 0; i < array.length; i++) {  
            reversedArray[i] = array[array.length - 1 - i];  
        }  
        return reversedArray;  
    }  
    private boolean isSameBoard(int[][] board1, int[][] board2) {  
        for (int i = 0; i < BOARD\_SIZE; i++) {  
            for (int j = 0; j < BOARD\_SIZE; j++) {  
                if (board1[i][j] != board2[i][j]) {  
                    return false;  
                }  
            }  
        }  
        return true;  
    }  
    private boolean isGameOver() {  
        if (hasEmptyCell()) {  
            return false;  
        }  
        int[][] tempBoard = new int[BOARD\_SIZE][BOARD\_SIZE];  
        for (int i = 0; i < BOARD\_SIZE; i++) {  
            System.arraycopy(board[i], 0, tempBoard[i], 0, BOARD\_SIZE);  
        }  
        return !moveTiles(tempBoard, "W") && !moveTiles(tempBoard, "A")  
                && !moveTiles(tempBoard, "S") && !moveTiles(tempBoard, "D");  
    }  
    private boolean hasEmptyCell() {  
        for (int[] row : board) {  
            for (int cell : row) {  
                if (cell == 0) {  
                    return true;  
                }  
            }  
        }  
        return false;  
    }  
    private boolean hasReached2048() {  
        for (int[] row : board) {  
            for (int cell : row) {  
                if (cell == 2048) {  
                    return true;  
                }  
            }  
        }  
        return false;  
    }  
    @Override  
    public void keyTyped(KeyEvent e) {  
    }  
    @Override  
    public void keyReleased(KeyEvent e) {  
    }  
    public static void main(String[] args) {  
        SwingUtilities.invokeLater(() -> {  
            Main2048 game = new Main2048();  
            game.setVisible(true);  
        });  
    }  
}