```java
/*
 * Last Update: 24/1/2025 - 10:28 PM
 *
 * This class handles the rotation, resizing, and flipping of icons
 * and tiles on the application.
 *
 * This class follows the View pattern in the MVC pattern.
 *
 * It's responsible to:
 * 1. Rotate icons associated with the tiles
 * 2. Resize icons and ensure they fit dynamically with the board size
 * 3. Flipping the board for player perspective changes
 * 4. Manage tile rotation logic based on game rules and the
 * current player's turn.
 *
 * This class interacts with Tile and Board classes to modify
 * the visual representation of the game dynamically during
 * the play.
*/


//written by Lycia
import java.awt.*;
import javax.swing.*;
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;

public class Rotation {

    // Reference to the controller for managing game state
    GameControl gameControl;

    // Constructor to initialize the Rotation class with a GameControl instance
    public Rotation(GameControl gameControl) {
        this.gameControl = gameControl;
    }

    // Rotate the icon on a specified tile by 180 degrees. If its
    // rotation status is set to true.
    public void rotateIcon(Tile t) {
        if (t.getRotationStatus()) { // check if rotation is required
            Icon icon = t.getDefaultImg();
            if (icon instanceof ImageIcon) {
                ImageIcon imageIcon = (ImageIcon) icon;
                Image image = imageIcon.getImage();
                Image rotatedImage = rotateImage(image, 180); // Rotate by 180 degrees
                Image resizedImage = rotatedImage.getScaledInstance(100, 100,
                Image.SCALE_SMOOTH);
                t.setRotatedImg(new ImageIcon(resizedImage)); // set the rotated image as icon
                t.setIconAtTile(); // update the tile with the new icon
            }
        } else {
            t.setIconAtTile(); // use default icon if no rotation is required
        }
    }

    // Rotate the given image by a specified degree
    public Image rotateImage(Image image, int degrees) {
        if (image == null) {
            throw new IllegalArgumentException("Image cannot be null");
        }

        int width = image.getWidth(null);
        int height = image.getHeight(null);

```

```
65          // Validate dimensions
66          if (width <= 0 || height <= 0) {
67              throw new IllegalArgumentException("Width and height must be greater than 0");
68          }
69
70          BufferedImage rotatedImage = new BufferedImage(width, height,
            BufferedImage.TYPE_INT_ARGB);
71          Graphics2D g2d = rotatedImage.createGraphics();
72
73          AffineTransform at = new AffineTransform();
74          at.rotate(Math.toRadians(degrees), width / 2.0, height / 2.0); // Rotate around the
            center
75          g2d.setTransform(at);
76
77          g2d.drawImage(image, 0, 0, null);
78          g2d.dispose();
79
80          return rotatedImage;
81      }
82
83      // Resize the icon (default and rotated) on a specific tile
84      // to the specified width and height
85      public void resizeImages(int width, int height, int x, int y) {
86          if (Tile.tiles[x][y].getDefaultImg() != null) {
87              Image image = Tile.tiles[x][y].getDefaultImg().getImage();
88              Image resizedImage = image.getScaledInstance(width, height, Image.SCALE_SMOOTH);
89              Tile.tiles[x][y].setDefaultImg(new ImageIcon(resizedImage));
90              Tile.tiles[x][y].setIconAtTile();
91          }
92
93          if (Tile.tiles[x][y].getRotatedImg() != null) {
94              Image image = Tile.tiles[x][y].getRotatedImg().getImage();
95              Image resizedImage = image.getScaledInstance(width, height, Image.SCALE_SMOOTH);
96              Tile.tiles[x][y].setRotatedImg(new ImageIcon(resizedImage));
97              Tile.tiles[x][y].setIconAtTile();
98          }
99      }
100
101     // Resize icons on the board based on the size of the tiles
102     public void resizeToOriginal() {
103         for (int i = 0; i < Board.row; i++) {
104             for (int j = 0; j < Board.column; j++) {
105                 resizeImages(Tile.tiles[i][j].getWidth() / 2, Tile.tiles[i][j].getHeight(), i,
                    j);
106             }
107         }
108     }
109
110     //written by Maya
111     // Flip the board by rearranging the tiles to provide a different perspective.
112     public void flipBoard() {
113         Board.getBoard().getBoardPanel().removeAll();
114
115         if (!Board.getBoard().getRotationStatus()) {
116             // add tiles in reverse order
117             for (int i = Board.row - 1; i >= 0; i--) {
118                 for (int j = Board.column - 1; j >= 0; j--) {
119                     Board.getBoard().getBoardPanel().add(Tile.tiles[i][j]);
120                 }
121             }
122         } else {
123             // add tiles in regular order
124             for (int i = 0; i < Board.row; i++) {
125                 for (int j = 0; j < Board.column; j++) {
126                     Board.getBoard().getBoardPanel().add(Tile.tiles[i][j]);
```

```
                }
            }
        }

        Board.getBoard().getBoardPanel().revalidate(); // revalidate the panel to reflect changes
        Board.getBoard().getBoardPanel().repaint(); // repaint to update the UI
    }

    // Check if the icon on the tile needs to be rotated based on
    // the game rules and current turn
    public void checkRotation(Piece piece, char whoseTurn, int i, int j) {
        if (piece.getSide() != whoseTurn) { // If it's not the piece's turn, rotate the icon
            Tile.tiles[i][j].setTileRotationStatus(true);
            if (piece instanceof Ram) {
                Ram ram = (Ram) piece;

                // If Ram has switched direction, rotate to the opposite direction
                if (ram.hasDirectionSwitched()) {
                    // Reverse the rotation since the direction has switched

                    Tile.tiles[i][j].setTileRotationStatus(!Tile.tiles[i][j].getRotationStatus())
                    ;
                } else {
                    // Keep the existing rotation state based on current logic
                    if ((ram.getPosX() == 0 || ram.getPosX() == 5)) {
                        Tile.tiles[i][j].setTileRotationStatus(true); // Test
                    } else if (ram.getReversedB() || ram.getReversedR()) {
                        Tile.tiles[i][j].setTileRotationStatus(false);
                    }
                    if ((ram.getPosX() == 0 || ram.getPosX() == 5) && (ram.getReversedB() ||
                    ram.getReversedR())) {
                        Tile.tiles[i][j].setTileRotationStatus(true);
                    }
                }
            }
        } else {
            Tile.tiles[i][j].setTileRotationStatus(false);
            if (piece instanceof Ram) {
                Ram ram = (Ram) piece;

                // If Ram has switched direction, rotate to the opposite direction
                if (ram.hasDirectionSwitched()) {
                    // Reverse the rotation since the direction has switched

                    Tile.tiles[i][j].setTileRotationStatus(!Tile.tiles[i][j].getRotationStatus())
                    ;
                } else {
                    // Keep the existing rotation state based on current logic
                    if ((ram.getPosX() == 0 || ram.getPosX() == 5)) {
                        Tile.tiles[i][j].setTileRotationStatus(false);
                    } else if (ram.getReversedB() || ram.getReversedR()) {
                        Tile.tiles[i][j].setTileRotationStatus(false);
                    }
                    if ((ram.getPosX() == 0 || ram.getPosX() == 5) && (ram.getReversedB() ||
                    ram.getReversedR())) {
                        Tile.tiles[i][j].setTileRotationStatus(true);
                    }
                }
            }
        }
    }
}
```