```java
package Project3;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.imageio.ImageIO;
import javax.swing.*;

/*********************************************************************
 * CIS 163 Section 01
 * Project 3: Chess Game
 * ChessPanel Class
 *
 * This class is a public JPanel which handles the GUI for the chess
 * game.  Extends JPanel and implements ActionListener.
 *
 * @author George Fayette
 * @version 3/23/2019
 ********************************************************************/
public class ChessPanel extends JPanel implements ActionListener {

    /**
     * Private JButton array representing the chess board.
     */
    private JButton[][] board;

    /**
     * Private JButtons for undo, PvP mode, PvAI mode, strobe, and make
     * AI move.
     */
    private JButton undoButton, pvpButton, pvaiButton, aiMoveButton,
            strobeButton;

    /**
     * Private JLabel for showing the number of moves.
     */
    private JLabel movesLabel;

    /**
     * Private ChessModel representing the game.
     */
    private ChessModel model;

    /**
     * Private ImageIcon arrays for storing the images for white and
     * black pieces.
     */
    private ImageIcon[] whitePieces, blackPieces;

    /**
     * Private TileStrobes for the move to and move from locations.
     */
    private TileStrobe strobe, flash;

    /**
     * Private booleans representing the move from location being
     * selected, whether the game is being played against the AI, and
     * whether or not the strobe graphics are enabled.
     */
    private boolean firstTurnFlag, vsAI, strobeOn;

    /**
     * Private ints representing the coordinates that a chess piece is
     * moving from.
     */
    private int fromRow, fromCol;

    /**
     * Private final String array for storing the the game piece names.
     */
    private final String[] pieces =
            {"Pawn", "Rook", "Knight", "Bishop", "Queen", "King"};

    /**
     * Private final String array for storing the white piece images
     */
    private final String[] wFiles =
            {"resources/wPawn.png", "resources/wRook.png",
                    "resources/wKnight.png", "resources/wBishop.png",
                    "resources/wQueen.png", "resources/wKing.png"};

    /**
     * Private final String array for storing the black piece images
     */
    private final String[] bFiles =
            {"resources/bPawn.png", "resources/bRook.png",
                    "resources/bKnight.png", "resources/bBishop.png",
                    "resources/bQueen.png", "resources/bKing.png"};

    /*****************************************************************
     * Public default constructor.
     *****************************************************************/
    public ChessPanel() {
        model = new ChessModel();
        firstTurnFlag = true;
        vsAI = false;
        strobeOn = false;
        createIcons();

        JPanel boardPanel = new JPanel();
        boardPanel.setLayout(
                new GridLayout(model.numRows(), model.numColumns(), 1,
                        1));
        boardPanel.setPreferredSize(new Dimension(600, 600));
        board = new JButton[model.numRows()][model.numColumns()];
        for (int r = 0; r < model.numRows(); r++) {
            for (int c = 0; c < model.numColumns(); c++) {
                createButton(r, c);
                setBackGroundColor(r, c);
                boardPanel.add(board[r][c]);
            }
        }
        strobe = new TileStrobe(0, 0, 0);
        flash = new TileStrobe(0, 0, 0);

        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(2, 4, 1, 1));
        undoButton = new JButton("Undo");
        undoButton.addActionListener(this);
        pvpButton = new JButton("PvP");
        pvpButton.addActionListener(this);
        pvaiButton = new JButton("PvAI");
        pvaiButton.addActionListener(this);
        aiMoveButton = new JButton("AI Move");
        aiMoveButton.addActionListener(this);
        strobeButton = new JButton("Strobe");
        strobeButton.addActionListener(this);
        movesLabel = new JLabel("Moves: " + model.numMoves());
        buttonPanel.add(pvpButton);
        buttonPanel.add(pvaiButton);
        buttonPanel.add(aiMoveButton);
        buttonPanel.add(undoButton);
        buttonPanel.add(strobeButton);
        buttonPanel.add(movesLabel);

        add(new JLabel("CHESS"), BorderLayout.NORTH);
        add(boardPanel, BorderLayout.CENTER);
        add(buttonPanel, BorderLayout.SOUTH);
    }

    // Sets the background color for the board
    private void setBackGroundColor(int r, int c) {
        if ((c % 2 == 1 && r % 2 == 0) || (c % 2 == 0 && r % 2 == 1)) {
            board[r][c].setBackground(Color.LIGHT_GRAY);
        } else {
            board[r][c].setBackground(Color.WHITE);
        }
    }

    // Creates the JButtons for the board
    private void createButton(int r, int c) {
        if (model.pieceAt(r, c) == null) {
            board[r][c] = new JButton(null, null);
        } else {
            for (int i = 0; i < pieces.length; ++i) {
                if (model.pieceAt(r, c).type().equals(pieces[i])) {
                    if (model.pieceAt(r, c).player() == Player.WHITE) {
                        board[r][c] = new JButton(null, whitePieces[i]);
                    } else if (model.pieceAt(r, c).player() ==
                            Player.BLACK) {
                        board[r][c] = new JButton(null, blackPieces[i]);
                    }
                }
            }
        }
        board[r][c].addActionListener(this);
    }

    // Reads image files and stores in ImageIcon arrays
    private void createIcons() {
        whitePieces = new ImageIcon[pieces.length];
        blackPieces = new ImageIcon[(pieces.length)];
        try {
            for (int i = 0; i < pieces.length; ++i) {
                whitePieces[i] = new ImageIcon(ImageIO.read(
                        getClass().getResource(wFiles[i])));
                blackPieces[i] = new ImageIcon(ImageIO.read(
                        getClass().getResource(bFiles[i])));
            }
        } catch (Exception e) {
                System.out.println("Error creating icons");
            }
        }

        // method that updates the board
        private void displayBoard() {
            for (int r = 0; r < 8; r++) {
                for (int c = 0; c < 8; c++) {
                    if (model.pieceAt(r, c) == null) {
                        board[r][c].setIcon(null);
                    } else {
                        for (int i = 0; i < pieces.length; ++i) {
                            if (model.pieceAt(r, c).type()
                                    .equals(pieces[i])) {
                                if (model.pieceAt(r, c).player() ==
                                        Player.WHITE) {
                                    board[r][c].setIcon(whitePieces[i]);
                                } else if (model.pieceAt(r, c).player() ==
                                        Player.BLACK) {
                                    board[r][c].setIcon(blackPieces[i]);
                                }
                            }
                        }
                    }
                }
            }
            movesLabel.setText("Moves: " + model.numMoves());
            repaint();
        }

        /*****************************************************************
         * This method handles ActionEvents from the GUI elements.
         * @param event An ActionEvent from the GUI.
         *****************************************************************/
        public void actionPerformed(ActionEvent event) {
            for (int r = 0; r < model.numRows(); r++) {
                for (int c = 0; c < model.numColumns(); c++) {
                    if (board[r][c] == event.getSource()) {
                        // First click
                        if (firstTurnFlag) {
                            if (model.pieceAt(r, c) != null &&
                                    model.pieceAt(r, c).player() ==
                                            model.currentPlayer()) {
                                fromRow = r;
                                fromCol = c;
                                firstTurnFlag = false;
                                strobe = new TileStrobe(r, c, -1);
                            }
                            // If another piece is selected
                        } else if (model.pieceAt(r, c) != null &&
                                model.pieceAt(r, c).player() ==
                                        model.currentPlayer()) {
                            fromRow = r;
                            fromCol = c;
                            strobe.stop();
                            strobe = new TileStrobe(r, c, -1);
                            // Try the move
                        } else {
                            attemptMove(new Move(fromRow, fromCol, r, c));
                        }
                }
            }
        }

        if (undoButton == event.getSource()) {
            if (vsAI) {
                model.undo();
                checkStatus();
                model.undo();
            } else {
                model.undo();
            }

            firstTurnFlag = true;
            strobe.stop();
            displayBoard();
            checkStatus();
        }

        if (pvpButton == event.getSource()) {
            vsAI = false;
        }

        if (pvaiButton == event.getSource()) {
            vsAI = true;
            if (model.currentPlayer() == Player.BLACK) {
                model.AI();
                firstTurnFlag = true;
                strobe.stop();
                displayBoard();
                checkStatus();
            }
        }

        if (aiMoveButton == event.getSource()) {
            model.AI();
            firstTurnFlag = true;
            strobe.stop();
            displayBoard();
            checkStatus();
            if (model.currentPlayer() == Player.BLACK && vsAI) {
                model.AI();
                displayBoard();
                checkStatus();
            }
        }

        if (strobeButton == event.getSource()) {
            if (strobeOn) {
                strobeOn = false;
                for (int r = 0; r < model.numRows(); r++) {
                    for (int c = 0; c < model.numColumns(); c++) {
                        setBackGroundColor(r, c);
                    }
                }
            } else {
                strobeOn = true;
            }
        }
    }
```

```java


    // This method attempts to make a move chosen by the player
    private void attemptMove(Move m) {
        if (model.tryMove(m)) {
            firstTurnFlag = true;
            flash = new TileStrobe(m.toRow, m.toColumn, 51);
            strobe.stop();
            displayBoard();
            checkStatus();

            if (vsAI && model.currentPlayer() == Player.BLACK) {
                model.AI();
                displayBoard();
                checkStatus();
            }
        }
    }

    // This method checks the current game status and informs the
    // player or asks for input as necessary
    private void checkStatus() {
        model.updateStatus();

        if (model.GUIcode() == GUIcodes.UPGRADE) {
            if (vsAI && model.currentPlayer() == Player.WHITE) {
                model.upgradePawn("Queen");
            } else {
                String upgrade = JOptionPane.showInputDialog(null,
                        "Enter promotion type.\n" +
                                " R = Rook\nK = Knight\nB = " +
                                "Bishop\nDefault is Queen");
                if (upgrade == null) {
                    upgrade = "";
                }

                upgrade = upgrade.toLowerCase();
                if (upgrade.equals("r")) {
                    model.upgradePawn("Rook");
                } else if (upgrade.equals("k")) {
                    model.upgradePawn("Knight");
                } else if (upgrade.equals("b")) {
                    model.upgradePawn("Bishop");
                } else {
                    model.upgradePawn("Queen");
                }
            }
            displayBoard();
            model.updateStatus();
        }

        if (model.GUIcode() == GUIcodes.CHECKMATE) {
            flash.stop();
            flashBoard(153);

            if (model.currentPlayer() == Player.BLACK) {
                JOptionPane.showMessageDialog(null,
                        "CheckMate! White Wins!", "Hooray!",
                        JOptionPane.INFORMATION_MESSAGE,
                        whitePieces[5]);
            } else {
                JOptionPane.showMessageDialog(null,
                        "CheckMate! Black Wins!", "Hooray!",
                        JOptionPane.INFORMATION_MESSAGE,
                        blackPieces[5]);
            }
        } else if (model.GUIcode() == GUIcodes.DRAW) {
            flash.stop();
            flashBoard(153);

            JOptionPane
                    .showMessageDialog(null, "It's a Draw!!", "Draw!",
                            JOptionPane.INFORMATION_MESSAGE,
                            whitePieces[5]);

        } else if (model.GUIcode() == GUIcodes.IN_CHECK) {
            flash.stop();
            flashBoard(50);

            if (model.currentPlayer() == Player.BLACK) {
                JOptionPane
                        .showMessageDialog(null, "Black is in check!",
                                "Yikes!", JOptionPane.WARNING_MESSAGE,
                                blackPieces[0]);
            } else {
                JOptionPane
                        .showMessageDialog(null, "White is in check!",
                                "Yikes!", JOptionPane.WARNING_MESSAGE,
                                whitePieces[0]);
            }
        }
    }

    // This method flashes all tiles on the board
    private void flashBoard(int ticks) {
        for (int r = 0; r < model.numRows(); ++r) {
            for (int c = 0; c < model.numColumns(); ++c) {
                flash = new TileStrobe(r, c, ticks);
            }
        }
    }

    /*****************************************************************
     * CIS 163 Section 01
     * Project 3: Chess Game
     * TileStrobe Class
     *
     * This class rapidly changes the color of a tile on the board to
     * produce a strobe effect
     *
     * @author George Fayette
     * @version 3/23/2019
     *****************************************************************/
    private class TileStrobe extends TimerTask {
        java.util.Timer timer;
        JButton StrobeButton;
        int bRow;
        int bCol;
        int tickCounter;
        int numTicks;

        // Default constructor, strobes the tile at location r,c for
        // given number of ticks
        private TileStrobe(int r, int c, int ticks) {
            bRow = r;
            bCol = c;
            tickCounter = 0;
            numTicks = ticks;
            StrobeButton = board[bRow][bCol];
            timer = new java.util.Timer(true);
            timer.scheduleAtFixedRate(this, 0, 15);
        }

        /*************************************************************
         * This method is executed every time the TimerTask is called.
         ************************************************************/
        public void run() {
            if (numTicks < 0 || tickCounter < numTicks) {
                if (strobeOn) {
                    StrobeButton.setBackground(
                            new Color(tickCounter * 5 % 256,
                                    tickCounter * 5 % 256,
                                    tickCounter * 5 % 256));
                    ++tickCounter;
                }
            } else {
                stop();
            }
        }

        // This method stops the strobe effect and resets the tile
        // background.
        private void stop() {
            timer.cancel();
            this.cancel();
            setBackGroundColor(bRow, bCol);
        }
    }
}
```