```java
package Pieces;

import Game.Board;
import Game.Colour;
import Game.Player;

/**
 * This class represents the King piece
 * @author E
 */
public class King extends Piece {

    public boolean canMove;
    private boolean hasMoved;

    public King(Colour colour) {
        super(PieceType.King, colour, Integer.MAX_VALUE);
        hasMoved = false;
    }

    @Override
    public int threats(Board board, int row, int column) {
        Piece[][] currentBoard = board.getBoard();
        Piece toExamine;
        int threatened = 0;
        if (row >= 1) {
            // top
            toExamine = currentBoard[row - 1][column];
            if (toExamine != null) {
                if (this.isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
            }
            if (column >= 1) {
                // top-left
                toExamine = currentBoard[row - 1][column - 1];
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        threatened += toExamine.weight;
                    }
                }
            }
            if (column <= 6) {
                // top-right
                toExamine = currentBoard[row - 1][column + 1];
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        threatened += toExamine.weight;
                    }
                }
            }
        }
        if (row <= 6) {
            // bottom
            toExamine = currentBoard[row + 1][column];
            if (toExamine != null) {
                if (this.isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
            }
            if (column >= 1) {
                // bottom-left
                toExamine = currentBoard[row + 1][column - 1];
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        threatened += toExamine.weight;
                    }
                }
            }
```

```
            if (column <= 6) {
                // bottom-right
                toExamine = currentBoard[row + 1][column + 1];
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        threatened += toExamine.weight;
                    }
                }
            }
        }
        if (column >= 1) {
            // left
            toExamine = currentBoard[row][column - 1];
            if (toExamine != null) {
                if (this.isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
            }
        }
        if (column <= 6) {
            // right
            toExamine = currentBoard[row][column + 1];
            if (toExamine != null) {
                if (this.isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
            }
        }
        return threatened;
    }

    @Override
    public int[][] attacks(Board board, int row, int column) {
        int[][] attacked = new int[8][8];
        int nextX, nextY;
        if (row >= 1) {
            // top
            nextX = row - 1;
            nextY = column;
            attacked[nextX][nextY]++;
            if (column >= 1) {
                // top-left
                nextX = row - 1;
                nextY = column - 1;
                attacked[nextX][nextY]++;
            }
            if (column <= 6) {
                // top-right
                nextX = row - 1;
                nextY = column + 1;
                attacked[nextX][nextY]++;
            }
        }
        if (row <= 6) {
            // bottom
            nextX = row + 1;
            nextY = column;
            attacked[nextX][nextY]++;
            if (column >= 1) {
                // bottom-left
                nextX = row + 1;
                nextY = column - 1;
                attacked[nextX][nextY]++;
            }
            if (column <= 6) {
                // bottom-right
                nextX = row + 1;
                nextY = column + 1;
                attacked[nextX][nextY]++;
```

```
            }
        }
        if (column >= 1) {
            // right
            nextX = row;
            nextY = column + 1;
            attacked[nextX][nextY]++;
        }
        if (column <= 6) {
            // left
            nextX = row;
            nextY = column - 1;
            attacked[nextX][nextY]++;
        }
        return attacked;
    }

    @Override
    public boolean[][] validMoves(Player opponent, Board board, int row, int column) {
        Piece[][] currentBoard = board.getBoard();
        Piece toExamine;
        // reset to false and check
        canMove = false;
        boolean[][] validPositions = new boolean[8][8];
        int nextX, nextY;
        if (row >= 1) {
            // top
            nextX = row - 1;
            nextY = column;
            toExamine = currentBoard[nextX][nextY];
            if ((toExamine == null | toExamine != null &&
            this.isOppositeColour(toExamine))
                    && this.isThreatened(board, nextX, nextY) == 0) {
                validPositions[nextX][nextY] = true;
                canMove = true;
            }
            if (column >= 1) {
                // top-left
                nextX = row - 1;
                nextY = column - 1;
                toExamine = currentBoard[nextX][nextY];
                if ((toExamine == null | toExamine != null &&
                this.isOppositeColour(toExamine))
                        && this.isThreatened(board, nextX, nextY) == 0) {
                    validPositions[nextX][nextY] = true;
                    canMove = true;
                }
            }
            if (column <= 6) {
                // top-right
                nextX = row - 1;
                nextY = column + 1;
                toExamine = currentBoard[nextX][nextY];
                if ((toExamine == null | toExamine != null &&
                this.isOppositeColour(toExamine))
                        && this.isThreatened(board, nextX, nextY) == 0) {
                    validPositions[nextX][nextY] = true;
                    canMove = true;
                }
            }
        }
        if (row <= 6) {
            // bottom
            nextX = row + 1;
            nextY = column;
            toExamine = currentBoard[nextX][nextY];
            if ((toExamine == null | toExamine != null &&
            this.isOppositeColour(toExamine))
                    && this.isThreatened(board, nextX, nextY) == 0) {
```

```
                validPositions[nextX][nextY] = true;
                canMove = true;
            }
            if (column >= 1) {
                // bottom-left
                nextX = row + 1;
                nextY = column - 1;
                toExamine = currentBoard[nextX][nextY];
                if ((toExamine == null | toExamine != null &&
                this.isOppositeColour(toExamine))
                        && this.isThreatened(board, nextX, nextY) == 0) {
                    validPositions[nextX][nextY] = true;
                    canMove = true;
                }
            }
            if (column <= 6) {
                // bottom-right
                nextX = row + 1;
                nextY = column + 1;
                toExamine = currentBoard[nextX][nextY];
                if ((toExamine == null | toExamine != null &&
                this.isOppositeColour(toExamine))
                        && this.isThreatened(board, nextX, nextY) == 0) {
                    validPositions[nextX][nextY] = true;
                    canMove = true;
                }
            }
            if (row >= 1) {
                // right
                nextX = row;
                nextY = column + 1;
                toExamine = currentBoard[nextX][nextY];
                if ((toExamine == null | toExamine != null &&
                this.isOppositeColour(toExamine))
                        && this.isThreatened(board, nextX, nextY) == 0) {
                    validPositions[nextX][nextY] = true;
                    canMove = true;
                }
            }
            if (column <= 6) {
                // left
                nextX = row;
                nextY = column - 1;
                toExamine = currentBoard[nextX][nextY];
                if ((toExamine == null | toExamine != null &&
                this.isOppositeColour(toExamine))
                        && this.isThreatened(board, nextX, nextY) == 0) {
                    validPositions[nextX][nextY] = true;
                    canMove = true;
                }
            }
        }
        // castle check
        int pos = colour == Colour.White ? 7 : 0;
        int[][] oppAttacks = opponent.getAttacks();
        if (column == 4 && !hasMoved && this.isThreatened(board, pos, 4) == 0) {
            // examine rooks
            // check queen side
            toExamine = currentBoard[pos][0];
            if (toExamine != null && toExamine.validSpecial()) {
                // check queen side
                if (oppAttacks[pos][column - 1] == 0
                        && oppAttacks[pos][column - 2] == 0) {
                    validPositions[pos][2] = true;
                    canMove = true;
                }
            }
            toExamine = currentBoard[pos][7];
            if (toExamine != null && toExamine.validSpecial()) {
```

```
                // check king side
                if (oppAttacks[pos][column + 1] == 0
                        && oppAttacks[pos][column + 2] == 0) {
                    validPositions[pos][6] = true;
                    canMove = true;
                }
            }
        }
        return validPositions;
    }

    @Override
    public boolean validSpecial() {
        return !hasMoved; // if it hasn't moved it can castle
    }

    @Override
    public void modifySpecial() {
        hasMoved = true;
    }

    @Override
    public String printToBoard() {
        return this.colour == Colour.White ? "\u2654" : "\u265A";
    }

    public String printToLog() {
        return "K";
    }

    public boolean castle(Rook rook) {
        return false; // need to check if rook has moved
    }

    @Override
    public boolean getCanMove() {
        return canMove;
    }
}

```