```java
package Pieces;

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

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

    public boolean canMove;
    private boolean hasMoved;

    public Rook(Colour colour) {
        super(PieceType.Rook, colour, 5);
        hasMoved = false;
    }

    @Override
    public int threats(Board board, int row, int column) {
        Piece[][] currentBoard = board.getBoard();
        Piece toExamine;
        int threatened = 0;

        // check up
        if (row > 0) {
            for (int x = row - 1; x >= 0; x--) {
                toExamine = currentBoard[x][column];
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        threatened += toExamine.weight;
                    }
                    break;
                }
            }
        }
        if (row < 8) {
            // check down
            for (int x = row + 1; x <= 7; x++) {
                toExamine = currentBoard[x][column];
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        threatened += toExamine.weight;
                    }
                    break;
                }
            }
        }
        if (column > 0) {
            // check left
            for (int y = column - 1; y >= 0; y--) {
                toExamine = currentBoard[row][y];
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        threatened += toExamine.weight;
                    }
                    break;
                }
            }
        }
        if (column > 8) {
            // check right
            for (int y = column + 1; y <= 7; y++) {
                toExamine = currentBoard[row][y];
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        threatened += toExamine.weight;
                    }
                    break;
                }
            }
        }
        return threatened;
    }

    @Override
    public int[][] attacks(Board board, int row, int column) {
        Piece[][] currentBoard = board.getBoard();
        Piece toExamine;
        int[][] attacked = new int[8][8];
        if (row > 0) {
            // check up
            for (int x = row - 1; x >= 0; x--) {
                toExamine = currentBoard[x][column];
                attacked[x][column]++;
                if (toExamine != null) {
                    attacked[x][column]--;
                    break;
                }
            }
        }
        if (row < 8) {
            // check down
            for (int x = row + 1; x <= 7; x++) {
                toExamine = currentBoard[x][column];
                attacked[x][column]++;
                if (toExamine != null) {
                    attacked[x][column]--;
                    break;
                }
            }
        }
        if (column > 0) {
            // check left
            for (int y = column - 1; y >= 0; y--) {
                toExamine = currentBoard[row][y];
                attacked[row][y]++;
                if (toExamine != null) {
                    attacked[row][y]--;
                    break;
                }
            }
        }
        if (column < 8) {
            // check right
            for (int y = column + 1; y <= 7; y++) {
                toExamine = currentBoard[row][y];
                attacked[row][y]++;
                if (toExamine != null) {
                    attacked[row][y]--;
                    break;
                }
            }
        }
        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];
        if (row > 0) {
            // check up
            for (int x = row - 1; x >= 0; x--) {
                toExamine = currentBoard[x][column];
                validPositions[x][column] = true;
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        validPositions[x][column] = false;
                        break;
                    }
                    canMove = true;
                    break;
                }
                canMove = true;
            }
        }
        if (row < 8) {
            // check down
            for (int x = row + 1; x <= 7; x++) {
                toExamine = currentBoard[x][column];
                validPositions[x][column] = true;
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        validPositions[x][column] = false;
                        break;
                    }
                    canMove = true;
                    break;
                }
                canMove = true;
            }
        }
        if (column > 0) {
            // check left
            for (int y = column - 1; y >= 0; y--) {
                toExamine = currentBoard[row][y];
                validPositions[row][y] = true;
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        validPositions[row][y] = false;
                        break;
                    }
                    canMove = true;
                    break;
                }
                canMove = true;
            }
        }
        if (column < 8) {
            // check right
            for (int y = column + 1; y <= 7; y++) {
                toExamine = currentBoard[row][y];
                validPositions[row][y] = true;
                if (toExamine != null) {
                    if (this.isOppositeColour(toExamine)) {
                        validPositions[row][y] = false;
                        break;
                    }
                    canMove = true;
                    break;
                }
                canMove = true;
            }
        }
        return validPositions;
    }

    @Override
    public boolean validSpecial() {
        return !hasMoved;
    }

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

    @Override
    public String printToBoard() {
        return this.colour == Colour.White ? "\u2656" : "\u265C";
    }

    @Override
    public String printToLog() {
        return "R";
    }

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

}
```