```java
package Pieces;

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

/**
 * This class represents the Bishop piece
 *
 * @author Param
 */
public class Bishop extends Piece {

    public boolean canMove;

    public Bishop(Colour colour) {
        super(PieceType.Bishop, colour, 3);
    }

    @Override
    public int threats(Board board, int row, int column) {
        Piece[][] currentBoard = board.getBoard();
        Piece toExamine;
        int threatened = 0;
        int posx = row;
        int posy = column;
        // diagonal top-left
        while (posx > 0 && posy > 0) {
            toExamine = currentBoard[posx--][posy--];
            if (toExamine != null) {
                if (this.isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
                break;
            }
        }
        posx = row;
        posy = column;
        // diagonal top-right
        while (posx > 0 && posy < 7) {
            toExamine = currentBoard[posx--][posy++];
            if (toExamine != null) {
                if (this.isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
                break;
            }
        }
        posx = row;
        posy = column;
        // diagonal bottom-left
        while (posx < 7 && posy > 0) {
            toExamine = currentBoard[posx++][posy--];
            if (toExamine != null) {
                if (this.isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
                break;
            }
        }
        posx = row;
        posy = column;
        // diagonal bottom-right
        while (posx < 7 && posy < 7) {
            toExamine = currentBoard[posx++][posy++];
            if (toExamine != null) {
                if (this.isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
```

```java
                    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];
            int posx = row;
            int posy = column;
            // diagonal top-left
            while (posx > 0 && posy > 0) {
                toExamine = currentBoard[posx--][posy--];
                attacked[posx][posy]++;
                if (toExamine != null) {
                    attacked[posx][posy]--;
                    break;
                }
            }
            posx = row;
            posy = column;
            // diagonal top-right
            while (posx > 0 && posy < 7) {
                toExamine = currentBoard[posx--][posy++];
                attacked[posx][posy]++;
                if (toExamine != null) {
                    attacked[posx][posy]--;
                    break;
                }
            }
            posx = row;
            posy = column;
            // diagonal bottom-left
            while (posx < 7 && posy > 0) {
                toExamine = currentBoard[posx++][posy--];
                attacked[posx][posy]++;
                if (toExamine != null) {
                    attacked[posx][posy]--;
                    break;
                }
            }
            posx = row;
            posy = column;
            // diagonal bottom-right
            while (posx < 7 && posy < 7) {
                toExamine = currentBoard[posx++][posy++];
                attacked[posx][posy]++;
                if (toExamine != null) {
                    attacked[posx][posy]--;
                    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];
            int posx = row;
            int posy = column;
            // diagonal top-left
            while (posx > 0 && posy > 0) {
```

```java
            toExamine = currentBoard[posx--][posy--];
            validPositions[posx][posy] = true;
            if (toExamine != null) {
                if (!this.isOppositeColour(toExamine)) {
                    validPositions[posx][posy] = false;
                    break;
                }
                canMove = true;
                break;
            }
            canMove = true;
        }
        posx = row;
        posy = column;
        // diagonal top-right
        while (posx > 0 && posy < 7) {
            toExamine = currentBoard[posx--][posy++];
            validPositions[posx][posy] = true;
            if (toExamine != null) {
                if (!this.isOppositeColour(toExamine)) {
                    validPositions[posx][posy] = false;
                    break;
                }
                canMove = true;
                break;
            }
            canMove = true;
        }
        posx = row;
        posy = column;
        // diagonal bottom-left
        while (posx < 7 && posy > 0) {
            toExamine = currentBoard[posx++][posy--];
            validPositions[posx][posy] = true;
            if (toExamine != null) {
                if (!this.isOppositeColour(toExamine)) {
                    validPositions[posx][posy] = false;
                    break;
                }
                canMove = true;
                break;
            }
            canMove = true;
        }
        posx = row;
        posy = column;
        // diagonal bottom-right
        while (posx < 7 && posy < 7) {
            toExamine = currentBoard[posx++][posy++];
            validPositions[posx][posy] = true;
            if (toExamine != null) {
                if (!this.isOppositeColour(toExamine)) {
                    validPositions[posx][posy] = false;
                    break;
                }
                canMove = true;
                break;
            }
            canMove = true;
        }
        return validPositions;
    }

    @Override
    public boolean validSpecial() {
        return false;
    }

    @Override
    public void modifySpecial() {
        // nothing
    }

    @Override
    public String printToBoard() {
        return this.colour == Colour.White ? "\u2657" : "\u265D";
    }

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

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

}
```