```java
package Pieces;

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

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

    public boolean canMove;
    public boolean hasMovedTwo;

    public Pawn(Colour colour) {
        super(PieceType.Pawn, colour, 1);
        hasMovedTwo = false;
    }

    @Override
    public int threats(Board board, int row, int column) {
        Piece[][] currentBoard = board.getBoard();
        Piece toExamine;
        int threatened = 0;
        if (colour == Colour.White && row >= 1) {
            if (column >= 1) {
                toExamine = currentBoard[row - 1][column - 1];
                if (toExamine != null && isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
            }
            if (column <= 6) {
                toExamine = currentBoard[row - 1][column + 1];
                if (toExamine != null && isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
            }
        } else if (colour == Colour.Black && row <= 6) {
            if (column >= 1) {
                toExamine = currentBoard[row + 1][column - 1];
                if (toExamine != null && isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
            }
            if (column <= 6) {
                toExamine = currentBoard[row + 1][column + 1];
                if (toExamine != null && isOppositeColour(toExamine)) {
                    threatened += toExamine.weight;
                }
            }
        }
        return threatened;
    }

    @Override
    public int[][] attacks(Board board, int row, int column) {
        int[][] attacked = new int[8][8];
        if (colour == Colour.White && row >= 1) {
            // check if there are pieces that can be taken
            if (column >= 1) {
                attacked[row - 1][column - 1]++;

            }
            if (column <= 6) {
                attacked[row - 1][column + 1]++;

            }
```

```
70          } else if (colour == Colour.Black && row <= 6) {
71              // check if there are pieces that can be taken
72              if (column >= 1) {
73                  attacked[row + 1][column - 1]++;
74
75              }
76              if (column <= 6) {
77                  attacked[row + 1][column + 1]++;
78
79              }
80          }
81          return attacked;
82      }
83
84      @Override
85      public boolean[][] validMoves(Player opponent, Board board, int row, int column) {
86          Piece[][] currentBoard = board.getBoard();
87          Piece toExamine;
88          // reset to false and check
89          canMove = false;
90          boolean[][] validPositions = new boolean[8][8];
91          if (colour == Colour.White && row >= 1) {
92              // check if it can move up
93              toExamine = currentBoard[row - 1][column];
94              if (toExamine == null) {
95                  validPositions[row - 1][column] = true;
96                  canMove = true;
97              }
98              // check if there are pieces that can be taken
99              if (column >= 1) {
100                 // check if there are pieces that can be taken
101                 toExamine = currentBoard[row - 1][column - 1];
102                 if (toExamine != null && isOppositeColour(toExamine)) {
103                     validPositions[row - 1][column - 1] = true;
104                     canMove = true;
105                 }
106             }
107             if (column <= 6) {
108                 toExamine = currentBoard[row - 1][column + 1];
109                 if (toExamine != null && isOppositeColour(toExamine)) {
110                     validPositions[row - 1][column + 1] = true;
111                     canMove = true;
112                 }
113             }
114             // check if it can move up 2
115             if (row == 6) {
116                 toExamine = currentBoard[row - 2][column];
117                 if (toExamine == null && currentBoard[row - 1][column] == null) {
118                     validPositions[row - 2][column] = true;
119                     canMove = true;
120                 }
121             } // check if it can perform en passant
122             else if (row == 3) {
123                 // use opponent info
124                 Piece lastMoved = opponent.getLastMoved();
125                 if (lastMoved.piece == PieceType.Pawn && lastMoved.validSpecial()) {
126                     //checks if my pawn is in right position and moves to right space
127                     if (opponent.getLastC() == column - 1
128                             || opponent.getLastC() == column + 1) {
129                         validPositions[row - 1][opponent.getLastC()] = true;
130                         canMove = true;
131                     }
132                 }
133             }
134         } else if (colour == Colour.Black && row <= 6) {
135             // check if it can move up
136             toExamine = currentBoard[row + 1][column];
137             if (toExamine == null) {
138                 validPositions[row + 1][column] = true;
```

```
                canMove = true;
            }
            // check if there are pieces that can be taken
            if (column >= 1) {
                toExamine = currentBoard[row + 1][column - 1];
                if (toExamine != null && isOppositeColour(toExamine)) {
                    validPositions[row + 1][column - 1] = true;
                    canMove = true;
                }
            }
            if (column <= 6) {
                // check if there are pieces that can be taken
                toExamine = currentBoard[row + 1][column + 1];
                if (toExamine != null && isOppositeColour(toExamine)) {
                    validPositions[row + 1][column + 1] = true;
                    canMove = true;
                }
            }
            // check if it can move up 2
            if (row == 1) {
                toExamine = currentBoard[row + 2][column];
                if (toExamine == null && currentBoard[row + 1][column] == null) {
                    validPositions[row + 2][column] = true;
                    canMove = true;
                }
            } // check if it can perform en passant
            else if (row == 4) {
                // use opponent info
                Piece lastMoved = opponent.getLastMoved();
                if (lastMoved.piece == PieceType.Pawn && lastMoved.validSpecial()) {
                    //checks if my pawn is in right position and moves to right space
                    if (opponent.getLastC() == column - 1
                            || opponent.getLastC() == column + 1) {
                        validPositions[row + 1][opponent.getLastC()] = true;
                        canMove = true;
                    }
                }
            }
        }
        return validPositions;
    }

    @Override
    public boolean validSpecial() {
        // has moved two, thus can be en passant (other conditions elsewhere)
        return hasMovedTwo;
    }

    @Override
    public void modifySpecial() {
        // only call method if piece confirmed to move two
        hasMovedTwo = true;
    }

    @Override
    public String printToBoard() {
        return this.colour == Colour.White ? "\u2659" : "\u265F";
    }

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

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