```
/*
   Filename:        GomokoGame.java
   Programmer:  WHALLEY Whitney Nicole
   Student Number: 240315876
   Class: SE1A
   Description: Create a multiplication table
   LINK TO GITHUB - https://github.com/whitneynciole/gomoko/
blob/237d727d3ed9e067087a317cc38220ef992f17f4/GomokoGame.java
 */

import java.util.Scanner;

public class GomokoGame {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Declare array for printing game table
        int[][] grid = new int[10][10];

        //initalize game grid array
        for (int row = 0; row < grid.length; row++) {
            for (int col = 0; col < 10; col++) {
                grid[row][col] = 0;
            }
        }

        // Display the initial grid
        displayGrid(grid);

        //Initalize players
        int currentPlayer = 1;
        boolean gameFinished = false;

        // play game
        while (gameFinished == false) {
            // Getting user input
            System.out.println("Player " + currentPlayer + "'s
turn.");
            System.out.print("Enter row and column (e.g., 0 1):
");
            int row = input.nextInt();
            int col = input.nextInt();

            // Checking if the move is valid or not
```

```
            boolean validMove = checkValidMove(grid, row, col);

            // Place the move on the board if the move is valid.
If it is false, allow player to enter a move again
            if (validMove) {
                // Placing the move on the board
                place(grid, row, col, currentPlayer);

                // Show the current board now
                displayGrid(grid);

                // Checking if the game has ended because there
was a winner
                gameFinished = checkGameWinner(currentPlayer,
grid);

                // If the game finished because of a winner
                if (gameFinished) {
                    System.out.println("Player " + currentPlayer
+ " wins!");
                } else {
                    // If no winner yet, check also if the board
is full
                    gameFinished = checkTie(grid);

                    // if the game finished because of a tie
                    if (gameFinished) {
                        System.out.println("Draw!");
                    } else {
                        // continue the game. now switch to user
2
                        currentPlayer = currentPlayer % 2 + 1;
                    }
                }
            }
        }
    }

    // Method to check if move is valid: if its within rage/free
space return true otherwise return false
    public static boolean checkValidMove(int[][] grid, int row,
int col) {
        if (row < 0 || row >= 10 || col < 0 || col >= 10) {
            System.out.println("Out of range! Input again.");
            return false;
        } else if (grid[row][col] != 0) {
```

```
            System.out.println("Invalid move. Try again.");
            return false;
        } else {
            return true;
        }
    }

    // Place the move of the current player
    public static void place(int[][] grid, int row, int col, int
currentPlayer) {
        grid[row][col] = currentPlayer;
    }

    // method to outprint the grid
    public static void displayGrid(int[][] grid) {
        for (int row = 0; row < grid.length; row++) {
            System.out.print(row + "| ");
            for (int col = 0; col < 10; col++) {
                System.out.print(grid[row][col] + " ");
            }
            System.out.println();
        }
        System.out.println("+------------------------------");
        System.out.println("   " + "0 1 2 3 4 5 6 7 8 9");
    }

    //method to check for winning
    public static boolean checkGameWinner(int currentPlayer, int
grid[][]) {
        // four across
        for (int row = 0; row < grid.length; row++) {
            for (int col = 0; col < 10 - 3; col++) {
                if (grid[row][col] == currentPlayer
                        && grid[row][col + 1] == currentPlayer
                        && grid[row][col + 2] == currentPlayer
                        && grid[row][col + 3] == currentPlayer)
{
                    return true;
                }
            }
        }

        // four up and down
        for(int row = 0; row < grid.length - 3; row++){
            for(int col = 0; col < 10; col++){
                if (grid[row][col] == currentPlayer &&
```

```
                    grid[row+1][col] == currentPlayer &&
                    grid[row+2][col] == currentPlayer &&
                    grid[row+3][col] == currentPlayer){
                    return true;
                }
            }
        }

        // four diagonal /
        for(int row = 3; row < grid.length; row++){
            for(int col = 0; col < 10 - 3; col++){
                if (grid[row][col] == currentPlayer   &&
                    grid[row-1][col+1] == currentPlayer &&
                    grid[row-2][col+2] == currentPlayer &&
                    grid[row-3][col+3] == currentPlayer){
                    return true;
                }
            }
        }

        // four diagonal \
        for(int row = 0; row < grid.length - 3; row++){
            for(int col = 0; col < 10 - 3; col++){
                if (grid[row][col] == currentPlayer   &&
                    grid[row+1][col+1] == currentPlayer &&
                    grid[row+2][col+2] == currentPlayer &&
                    grid[row+3][col+3] == currentPlayer){
                    return true;
                }
            }
        }

        // No 4 in a row found, no winner yet
        return false;
    }

    // method to check for a tie
    public static boolean checkTie(int grid[][]) {
        for(int row = 3; row < grid.length; row++){
            for(int col = 0; col < 10 - 3; col++){
                // If any of the grid is empty, then there is no
tie yet
                if (grid[row][col] == 0) {
                    return false;
                }
            }
```

```
        }

        // If all of the grid is filled, then the game is a tie
        return true;
    }
}
```