Tic-Tac-Toe Game:

Create a simple two-player console-based game.

A console-based version of the classic two-player game. Players take turns marking spaces on a 3x3 grid, and the first player to get three marks in a row (horizontally, vertically, or diagonally) wins.

Input: Player move (row and column) on a 3x3 grid.
Output: Updated game board and game status (win/draw/continue).

Example:

- Input: Player 1 selects row 1, column 2
- Output: Updated grid, "Player 1 wins" or "Continue playing"

Solution 1: Tic-Tac-Toe with Simple Array and Loops

Code:

```
import java.util.Scanner;

public class TicTacToeSimple {
    // Declare a 3x3 array to represent the game board
    static char[][] board = new char[3][3];

    public static void main(String[] args) {
        // Initialize the board with empty spaces
        initializeBoard();

        // Variable to keep track of the current player ('X' or 'O')
        char currentPlayer = 'X';

        // Variable to check if the game is won or drawn
        boolean gameWon = false;
        boolean draw = false;

        // Create a Scanner object for input
        Scanner scanner = new Scanner(System.in);

        // Main game loop, runs until a player wins or it's a draw
        while (!gameWon && !draw) {
```

```java
            printBoard();  // Display the board
            System.out.println("Player " + currentPlayer + "'s turn. Enter

            // Get player input for row and column
            int row = scanner.nextInt() - 1;
            int col = scanner.nextInt() - 1;

            // Check if the selected cell is empty
            if (board[row][col] == ' ') {
                // Place the current player's mark on the board
                board[row][col] = currentPlayer;

                // Check if the current player has won the game
                gameWon = checkWin(currentPlayer);

                if (!gameWon) {
                    // Check if the board is full (draw)
                    draw = checkDraw();

                    // Switch the player if the game isn't won
                    currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
                }
            } else {
                System.out.println("Cell is already occupied! Try again.");
            }
        }

        // Display the final board
        printBoard();

        // Display game result
        if (gameWon) {
            System.out.println("Player " + currentPlayer + " wins!");
        } else {
            System.out.println("It's a draw!");
        }

        // Close the scanner
        scanner.close();
    }
```

```java
    // Function to initialize the board with empty spaces
    public static void initializeBoard() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                board[i][j] = ' ';
            }
        }
    }

    // Function to print the current state of the board
    public static void printBoard() {
        System.out.println("  1 2 3");
        for (int i = 0; i < 3; i++) {
            System.out.print((i + 1) + " ");
            for (int j = 0; j < 3; j++) {
                System.out.print(board[i][j]);
                if (j < 2) System.out.print("|");
            }
            System.out.println();
            if (i < 2) System.out.println("  -----");
        }
    }

    // Function to check if the current player has won
    public static boolean checkWin(char player) {
        // Check rows, columns, and diagonals
        for (int i = 0; i < 3; i++) {
            if (board[i][0] == player && board[i][1] == player && board[i]|
                return true;
            if (board[0][i] == player && board[1][i] == player && board[2]|
                return true;
        }
        if (board[0][0] == player && board[1][1] == player && board[2][2] =
            return true;
        if (board[0][2] == player && board[1][1] == player && board[2][0] =
            return true;
        return false;
    }
```

```java
        // Function to check if the board is full (indicating a draw)
        public static boolean checkDraw() {
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {
                    if (board[i][j] == ' ') {
                        return false;  // If any cell is empty, it's not a draw
                    }
                }
            }
            return true;  // If no empty cells, it's a draw
        }
    }
```

Output:

```
1 2 3
1  | |
  -----
2  | |
  -----
3  | |
Player X's turn. Enter row and column (1-3):
 1
 1
  1 2 3
1 X| |
  -----
2  | |
  -----
3  | |
Player O's turn. Enter row and column (1-3):
 1
 2
  1 2 3
1 X|O|
  -----
2  | |
  -----
3  | |
Player X's turn. Enter row and column (1-3):
 3
 3
  1 2 3
1 X|O|
  -----
2  | |
  -----
```

```
3  | |X
Player O's turn. Enter row and column (1-3):
 3
 2
  1 2 3
1 X|O|
  -----
2  | |
  -----
3  |O|X
Player X's turn. Enter row and column (1-3):
 2
 2
  1 2 3
1 X|O|
  -----
2  |X|
  -----
3  |O|X
Player X wins!
```

Explanation :

- Game Board**:** The board is represented by a 2D array, initialized with empty spaces.
- **Player Input:** Players input the row and column to place their mark.
- **Win/Draw Conditions:** The game checks for a win by rows, columns, or diagonals. It also checks if all spaces are filled (indicating a draw).
- **Switching Players:** The game alternates between 'X' and 'O'.
- **Loop:** The game loop runs until either a player wins or the game ends in a draw.

Solution 2: Using Object-Oriented Approach

Key points:

- **Game Board Initialization:** The board is initialized as a 3x3 grid, filled with empty spaces.
- **Player Moves:** The user inputs row and column values, and the game updates the board accordingly if the input is valid.
- **Win Condition:** After every move, the game checks if the current player has completed a row, column, or diagonal.
- **Draw Condition:** If the board is full and no one wins, the game is a draw.
- **Switching Players:** After every valid move, the current player switches between 'X' and 'O'.

- **Game Restart:** After a game ends, players can choose to restart the game with an empty board or end the session.
- **Statistics:** The game keeps track of how many games each player has won and the number of draws.
- **Input Validation:** Invalid inputs (e.g., selecting already occupied spots or entering out-of-bound values) are handled, and the player is prompted to re-enter their move.

Code:

TicTacToe.java

```
import java.util.Scanner;

class TicTacToe {
    private char[][] board;
    private char currentPlayer;
    private int playerXWins;
    private int playerOWins;
    private int draws;

    // Constructor to initialize the board and set the starting player
    public TicTacToe() {
        board = new char[3][3];
        currentPlayer = 'X';
        initializeBoard();
        playerXWins = 0;  // Track Player X wins
        playerOWins = 0;  // Track Player O wins
        draws = 0;        // Track draws
    }

    // Initialize the board with empty spaces
    private void initializeBoard() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                board[i][j] = ' ';
            }
        }
    }

    // Display the current state of the board
    public void printBoard() {
```

```
        System.out.println("  1 2 3");
        for (int i = 0; i < 3; i++) {
            System.out.print((i + 1) + " ");
            for (int j = 0; j < 3; j++) {
                System.out.print(board[i][j]);
                if (j < 2) System.out.print("|");
            }
            System.out.println();
            if (i < 2) System.out.println("  -----");
        }
    }

    // Switch the current player between 'X' and 'O'
    public void switchPlayer() {
        currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
    }

    // Handle player input and update the board
    public boolean makeMove(int row, int col) {
        if (row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] =
            board[row][col] = currentPlayer;
            return true;
        }
        return false;
    }

    // Check if the current player has won the game
    public boolean checkWin() {
        for (int i = 0; i < 3; i++) {
            if (board[i][0] == currentPlayer && board[i][1] == currentPlaye
                return true;
            if (board[0][i] == currentPlayer && board[1][i] == currentPlaye
                return true;
        }
        if (board[0][0] == currentPlayer && board[1][1] == currentPlayer &&
            return true;
        if (board[0][2] == currentPlayer && board[1][1] == currentPlayer &&
            return true;
        return false;
    }
```

```java
// Check if the board is full (indicating a draw)
public boolean checkDraw() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
                return false;
            }
        }
    }
    return true;
}

// Get the current player
public char getCurrentPlayer() {
    return currentPlayer;
}

// Update the win count for the current player
public void updateWinCount() {
    if (currentPlayer == 'X') {
        playerXWins++;
    } else {
        playerOWins++;
    }
}

// Increment the draw count
public void updateDrawCount() {
    draws++;
}

// Display the game statistics
public void displayStatistics() {
    System.out.println("Player X Wins: " + playerXWins);
    System.out.println("Player O Wins: " + playerOWins);
    System.out.println("Draws: " + draws);
}

// Reset the board for a new game
```

```
        public void resetBoard() {
            initializeBoard();
            currentPlayer = 'X';  // Player X always starts the new game
        }
    }
```

Explanation:

- **Class Definition:**
  - The TicTacToe class manages the game's logic, including the board, player turns, win checks, and statistics.
- **Fields:**
  - board: A 3x3 char array representing the game board.
  - currentPlayer: Tracks the current player ('X' or 'O').
  - playerXWins, playerOWins, draws: Integers to track the number of wins for each player and the number of draws.
- **Constructor:**
  - Initializes the board, sets the starting player to 'X', and resets the win/draw counters.
- **initializeBoard():**
  - Fills the board with empty spaces (' ') to prepare it for gameplay.
- **printBoard():**
  - Displays the current state of the board in a 3x3 grid format with row and column numbers for easy reference.
- **switchPlayer():**
  - Switches the current player between 'X' and 'O' after each turn.
- **makeMove(int row, int col):**
  - Places the current player's mark on the specified row and column, if the spot is valid and unoccupied.
  - Returns true if the move is valid, otherwise false.
- **checkWin():**
  - Checks all possible winning conditions (rows, columns, and diagonals) to determine if the current player has won.
  - Returns true if there is a win, otherwise false.

- **checkDraw():**
  - Checks if the board is full and no moves are possible, indicating a draw.
  - Returns true if the game is a draw, otherwise false.
- **getCurrentPlayer():**
  - Returns the current player ('X' or 'O').
- **updateWinCount():**
  - Increments the win count for the current player (either playerXWins or playerOWins).
- **updateDrawCount():**
  - Increments the draw count (draws).
- **displayStatistics():**
  - Displays the current statistics: number of wins for Player X, Player O, and the number of draws.
- **resetBoard():**
  - Resets the board for a new game, clearing all spaces and setting the starting player to 'X'.

TicTacToeOOP.java

```
import java.util.Scanner;
public class TicTacToeOOP {
    public static void main(String[] args) {
        TicTacToe game = new TicTacToe(); // Create a new TicTacToe game ob
        Scanner scanner = new Scanner(System.in);
        boolean playAgain = true;

        // Main game loop (for multiple games)
        while (playAgain) {
            boolean gameWon = false;
            boolean draw = false;

            // Game loop for a single game
            while (!gameWon && !draw) {
                game.printBoard();  // Display the game board
                System.out.println("Player " + game.getCurrentPlayer() + "

                // Get player input for row and column
                int row = scanner.nextInt() - 1;
```

```
                int col = scanner.nextInt() - 1;

                // Make the move and check for validity
                if (game.makeMove(row, col)) {
                    gameWon = game.checkWin();
                    if (!gameWon) {
                        draw = game.checkDraw();
                        if (!draw) {
                            game.switchPlayer();  // Switch player if game
                        } else {
                            System.out.println("It's a draw!");
                            game.updateDrawCount();  // Increment the draw
                        }
                    } else {
                        game.printBoard();
                        System.out.println("Player " + game.getCurrentPlaye
                        game.updateWinCount();  // Increment win count for
                    }
                } else {
                    System.out.println("Invalid move. Try again.");
                }
            }

            // Display the game statistics
            game.displayStatistics();

            // Ask if the players want to play again
            System.out.println("Do you want to play again? (yes/no):");
            String response = scanner.next();

            if (response.equalsIgnoreCase("yes")) {
                game.resetBoard();  // Reset the board for a new game
            } else {
                playAgain = false;  // Exit the game loop
            }
        }

        System.out.println("Thanks for playing!");
        scanner.close();
    }
```

```
    }
```

Explanation:

- **Imports:**
  - java.util.Scanner: Used to capture user input from the console.
- **main() Method:**
  - The entry point of the program where the game logic is executed.
- **Game Initialization:**
  - Creates a TicTacToe object named game.
  - Initializes a Scanner object to read user input.
- **Main Game Loop (while (playAgain)):**
  - Controls whether the players want to play another game.
  - Runs continuously until the players choose not to play again.
- **Single Game Loop:**
  - Repeats until either a player wins or the game ends in a draw.
  - Prompts the current player to enter row and column coordinates for their move.
- **Input Handling:**
  - Player enters row and column (1-3), which are converted to 0-indexed values.
  - Checks if the move is valid using game.makeMove().
- **Move Validation:**
  - **If the move is valid:**
    - Calls game.checkWin() to see if the current player has won.
    - If no one has won, calls game.checkDraw() to see if the game is a draw.
    - If the game is ongoing, switches the player using game.switchPlayer().
- **Winning Condition:**
  - If a player wins, the board is displayed, and the current player is declared the winner.
  - Increments the win count for the winner using game.updateWinCount().
- **Draw Condition:**

    - If the game is a draw, displays a message and increments the draw count with game.updateDrawCount().
- **Display Statistics:**
    - After each game, game.displayStatistics() shows the win and draw counts.
- **Play Again Prompt:**
    - Asks the players if they want to play another game.
    - If they answer "yes", the board is reset using game.resetBoard().
    - If "no", the loop ends, and the game concludes.
- **Game Conclusion:**
    - Displays a thank-you message and closes the Scanner object.

Output:

```
1 2 3
1  | |
  -----
2  | |
  -----
3  | |
Player X's turn. Enter row and column (1-3):
1
1
  1 2 3
1 X| |
  -----
2  | |
  -----
3  | |
Player O's turn. Enter row and column (1-3):
1
3
  1 2 3
1 X| |O
  -----
2  | |
  -----
3  | |
Player X's turn. Enter row and column (1-3):
2
1
  1 2 3
1 X| |O
  -----
2 X| |
```

```
  -----
3  | |
Player O's turn. Enter row and column (1-3):
3
1
  1 2 3
1 X| |O
  -----
2 X| |
  -----
3 O| |
Player X's turn. Enter row and column (1-3):
2
2
  1 2 3
1 X| |O
  -----
2 X|X|
  -----
3 O| |
Player O's turn. Enter row and column (1-3):
2
3
  1 2 3
1 X| |O
  -----
2 X|X|O
  -----
3 O| |
Player X's turn. Enter row and column (1-3):
1
2
  1 2 3
1 X|X|O
  -----
2 X|X|O
  -----
3 O| |
Player O's turn. Enter row and column (1-3):
3
3
  1 2 3
1 X|X|O
  -----
2 X|X|O
  -----
3 O| |O
Player O wins!
Player X Wins: 0
Player O Wins: 1
Draws: 0
```

```
Do you want to play again? (yes/no):
no
Thanks for playing!
```

Java Code Editor: