**REQUIREMENT**

Creating a Tic Tac Toe game in Java requires several components to meet the project requirements. Below is a sample breakdown of what could be expected for such a project

1. Project Overview

Objective: Develop a two-player Tic Tac Toe game where players can take turns to place their mark (either 'X' or 'O') on a 3x3 grid. The game should end when a player wins or when it's a draw.

2. Functional RequirementsPlayer Input:The game should allow two players to take turns, inputting their moves via the console.Players should enter a number (1 to 9) corresponding to positions on the 3x3 grid.Board Representation:The game board should be represented in a 3x3 grid (array) where each position is initially empty, and as the game progresses, each player's moves will be placed in the grid.Move Validation:The game should check that the player's move is valid:The move must be within the range of 1 to 9.The chosen cell must not already be occupied.

Game Flow:The game should alternate turns between Player 1 (X) and Player 2 (O).The game should check for a winner after each move, using the rules of Tic Tac Toe (3 consecutive marks in a row, column, or diagonal) The game should declare a winner when one player has won or end in a draw if all positions are filled without a winner.

Display the Board:After each move, the current state of the game board should be displayed to the players in a readable format Game End Conditions:The game ends when a player wins (three marks in a row, column, or diagonal).The game ends in a draw if the board is filled, and no player has won.Replay Option:After a game ends, prompt the players to either play again or quit.

3. Non-Functional RequirementsUser Interface:The game should have a console-based user interface. It will display the game board and take user input via the terminal/console.Error Handling:Proper error handling should be implemented for invalid input (e.g., entering a number outside of the valid range or entering a previously occupied cell).

4. Technical RequirementsProgramming Language:Use Object-Oriented Programming principles (encapsulation, inheritance, and polymorphism) to structure the game.Classes:Board Class: Handles the game board (3x3 grid) and checks for win conditions.

**UML Diagram**

**PROJECT**

-board: char[][]

-player: char

-gameOver: Boolean

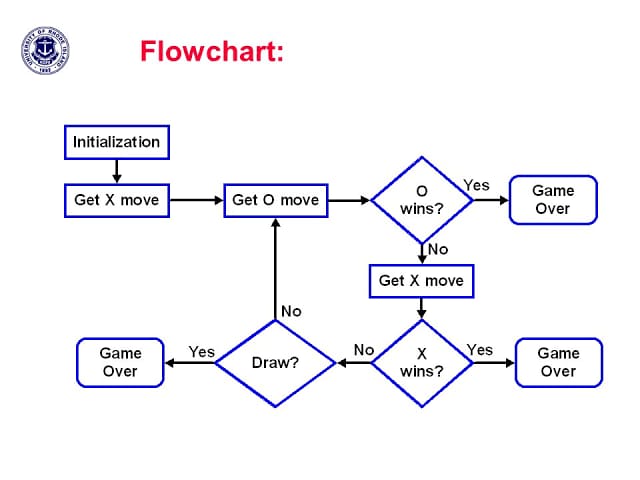
+ main(args: String[]): void

+ haveWon(board: char[][], player: char): boolean

+ printBoard(board: char[][]): void

+ isBoardFull(board: char[][]): Boolean

**FLOWCHART**



**SOURCE CODE**

import java.util.Scanner;

public class TicTacToe {

static char[][] board = {

{'1', '2', '3'},

{'4', '5', '6'},

{'7', '8', '9'}

};

static char currentPlayer = 'X';

public static void main(String[] args) {

int moves = 0;

boolean isWinner = false;

while (moves < 9 && !isWinner) {

printBoard();

playerMove();

moves++;

isWinner = checkWinner();

if (!isWinner) {

currentPlayer = (currentPlayer == 'X') ? 'O' : 'X'; // Switch player

}

}

printBoard();

if (isWinner) {

System.out.println("Player " + currentPlayer + " wins!");

} else {

System.out.println("It's a draw!");

}

}

public static void printBoard() {

System.out.println("Current Board:");

for (char[] row : board) {

for (char cell : row) {

System.out.print(cell + " ");

}

System.out.println();

}

}

public static void playerMove() {

Scanner scanner = new Scanner(System.in);

int move;

while (true) {

System.out.println("Player " + currentPlayer + ", enter your move (1-9): ");

move = scanner.nextInt();

if (move < 1 || move > 9) {

System.out.println("Invalid move! Please enter a number between 1 and 9.");

continue;

}

int row = (move - 1) / 3;

int col = (move - 1) % 3;

if (board[row][col] != 'X' && board[row][col] != 'O') {

board[row][col] = currentPlayer;

break;

} else {

System.out.println("Cell already taken! Choose another.");

}

}

}

public static boolean checkWinner() {

// Check rows, columns, and diagonals for a win

for (int i = 0; i < 3; i++) {

if ((board[i][0] == currentPlayer && board[i][1] == currentPlayer && board[i][2] == currentPlayer) ||

(board[0][i] == currentPlayer && board[1][i] == currentPlayer && board[2][i] == currentPlayer)) {

return true;

}

}

// Check diagonals

if ((board[0][0] == currentPlayer && board[1][1] == currentPlayer && board[2][2] == currentPlayer) ||

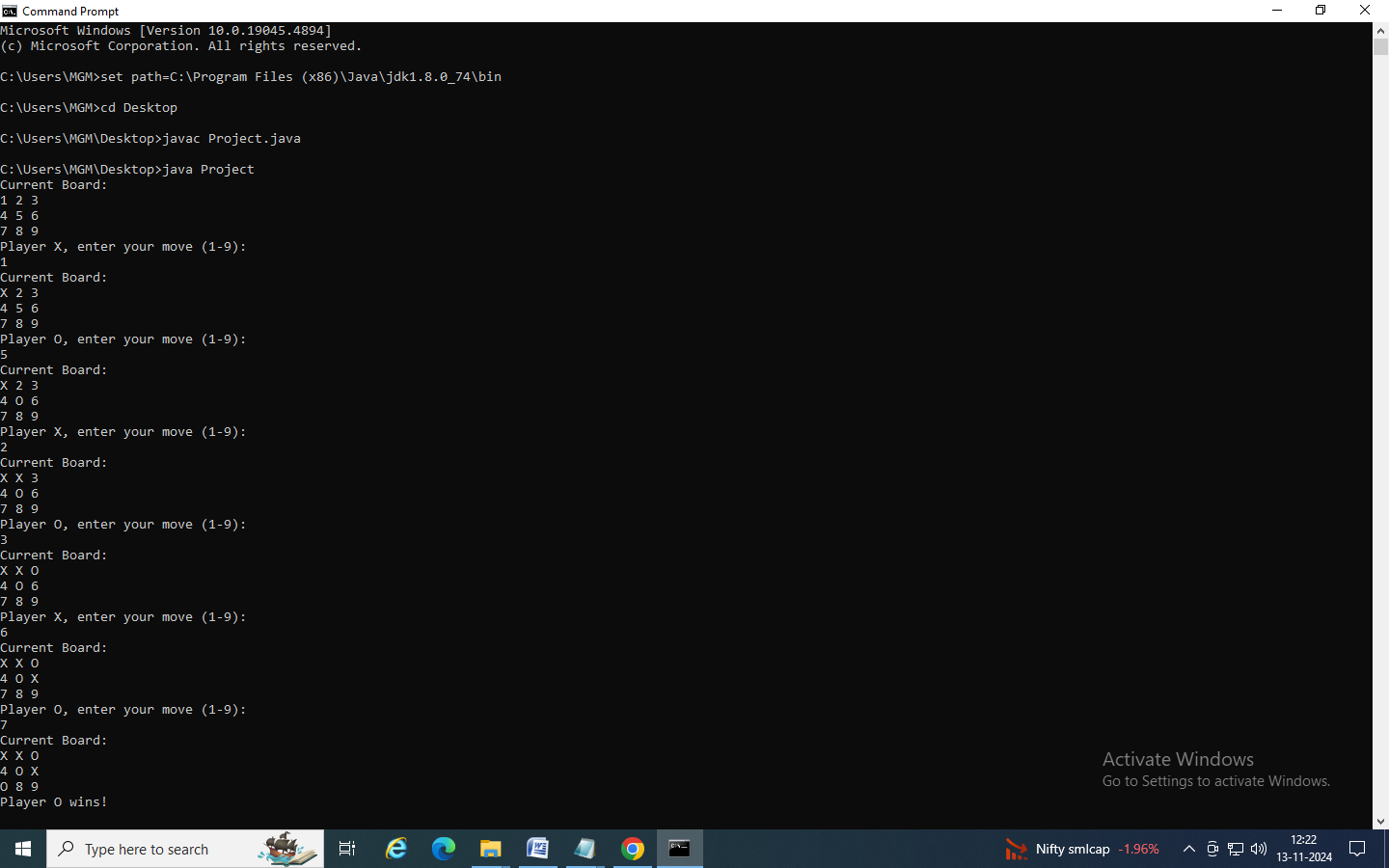
(board[0][2] == currentPlayer && board[1][1] == currentPlayer && board[2][0] == currentPlayer)) {

return true;

}

return false;

}



**CONCLUSION**

This Java program effectively implements a basic Tic-Tac-Toe game, showcasing several essential programming concepts. The game board is represented as a 2D character array, providing a clear structure that allows for easy updating and display. Through the use of loops and conditional statements, the program manages the game flow, including turn-taking, move validation, and win-condition checks. Key features include user input handling with the Scanner class, enabling interactive gameplay, and a modular design that organizes code into distinct functions like printBoard() and hasWon(), improving readability and isolating specific tasks. Overall, the program highlights the importance of clear code structure and modularity, both of which are vital skills in software development. By combining arrays, loops, conditionals, and user input handling, the code not only facilitates smooth gameplay but also provides foundational experience in building interactive applications. This project emphasizes problem-solving, design, debug, and implement game logic in Java.