I9rg National University

Peshawar

Name — Amir Sohail

ID — 16436

Department — BS(CS)

Assignmint No — 02

Submitted to — Ayub Sir

Question # 1

Tic Tac Toe game using 2 dimensional Array.

```
imPort jova . util . Scanner ;
Public class Tic Tac Toe
 {
Public static void main(String [] args)
 {
  Scanner in = new Scanner(System.in);
   int row, column;
   Char Player = 'X' ;
// create 2 dimensional array for TicTac Toe
   board
   Char [][] board = new char [3][3];
    Char ch = '1' ;
   for (int a=0 ; a<3 ; a++)
    {
   for (int b=0 ; b<3 ; b++)
   {
     board [a][b] = ch++;
    }
    }
```

```
displayBoard (board);
while (!winner (board) = true)
{
  // get input for row / column
System.out.PrintIn("Enter a row and column
  (0, 1, or 2); for Player" + Player + ":");
    row = in.nextInt;
    column = in.nextInt;
  while (board [row] [column] == 'X'
// board [row] [column] == 'O')
  {
System.out.PrintIn("This spot is occupied.
    Please try again.");
  }
  board [row] [column] = Player;
  displayBoard (board);
  if (winner (board)
  {
System.out.Println("Player" + Player +
  " is the winner!");
  }
// time to swap Player after
    each go.
```

```
if (Player == 'O')
 {
    Player = 'X';
 }
 else
 {
   Player = 'O';
 }
 if (winner (board) == false && ! hasfreesbace(board))
 {
System.out.Println("The is a draw. Please try again");
 }
 }
  in. close ();
  }
Public static void displayBoard(char[][] board)
 {
 for (int a=0; a< board.length; a++)
  {
    for(int b=0; board[a].length; b++;
    {
if(b== board[a].length -1 ) System.out.Print(board
  [a] [b];
```

```
else
System.out.Print(a][b] + "|");
}
System.out.println();
}
}
/*
Determines whetere the board is completely
occupied by X and O character.
 Param board the board to Search through
 return true if entire board is
Populated by X or O, false
otherwise. */
Public static boolean hasfreeSpace(char[][] board)
{
 for (int a=0; a<board; a++)
 {
for (int b=0; b<board[0].length;b++
{
 if (board[a][b] != 'O' || board[a][b] != 'X')
{
 return true;
 }
```

```
}
}
return false;
}
// method to determine there is winner
Public static boolean winner(char[][]board)
{
return is Horizontal win(board) || is vertical win(board)
|| is Diagonal win(board);
}
/* Determine if there is a winner by
checking each row for consecutive.
matching token:
return true if there is a win
horizontally, false otherise. */
Private static boolean isHorizontal win char[][](board)
{
for (int row=0; row<board.length; row++)
{
if (is win (board [row]))
    return true;
}
return false; }
```

```
Private static boolean is win(char[]
lineTop Process)
 {
    boolean foundwin = true;
   char Prevchar = '-';
for (char ~~choter~~ character : lineTop Process)
 {
   if (Prevchar == '-')
        prevchar = character;
if ('o' != character && 'x' character)
 {
   foundwin = false ;
   break ;
}
  else if (Prevchar != chacter)
{
   foundwin = false;
   break;
}
}
return foundwin;
}
```

```
/* Determine whether there is a winner
by checking column for consecutive
maching token.
return true if there is a vertical
winner, false otherwise. */
 Private static boolean inVerticalWin(char[][] board)
{
  char [] column = null;
 for (int col=0 ; col< board[0].length ; col++)
 {
  column = new char[broad].length ];
for (int row =0 ; row < column.length ; row++)
 {
   column [row] = board [row] [col];
 }
  if (isWin (column)
     return true;
 }
    return false;
  }
// return true if a diagonal
winner exists, false otherwise.
```

```
Private static boolean isDiagonalWin(char[][] board)
{
    int row = 0, col = 0;
    int cols = board.length;
    int rows = board[0].length;
    // Create a one-dimensional array
    to represent the diagonal.
    // of the row or column to set
    its size. If the grid is rectangular
    // a diagonal with always be the
    size of the lesser of two dimensions.
    int size = row < col ? row : col;
    Char[] diagonal = new Char[size];
    // Since we know the grid is a
    square we really could just check one
    // these - either row or col, but
    I left both in here
    anyway.
    while (row < row && col < col)
    {
        diagonal[col] = board[row][col];
        row++;
        Col++;
```

```
}
if (is win (diagonal)
  {
    return true;
  }
}
row = row - 1;
col = 0;
diagonal = new char [size];
while (row>0 && col < cols
  {
diagonal [col] = board [row][col];
row --;
col ++;
  }
return is win (diagonal);
}
}
```

The End