

**Tik –Tac-Toe**

**Prepared by:**

|  |  |  |
| --- | --- | --- |
| Name | ID | Group |
| Arsany Mousa Fathy Rezk | 6927 | 2 |
| Moustafa Ebrahim Hassan | 6955 | 2 |

**CLASSES DESCRIPTION**

**1)** GameSession

This class creates the game status by representing the new board each time the player chooses the playing position. It’s also responsible for handling the errors, and finally checking for the winning status.

It has 6 attributes 3 of type Player and 1 of type Board and a Boolean one called EndGame which decides whether the game still running or not as it will be illustrated later.

3 Player variables one for each player and a 1 for the player currently playing.

Th Board variable creates an instance of board where the game is constructed.

Methods used:

* Game session()

A constructor for creating the board, players, and the current player. By creating an instance of GameSession using the variables: **Player1, Player2, CurrentPlayer, Board.**

* Init()

It is responsible for taking the input from the player and decide whether the input is valid to be stored in the array or not.

It checks if the position chosen by the player is empty and not occupied by the opponent player. A simple if condition is made to check what does the position entered occupies either a player symbol or a (-) if true that it contains a player symbol then the function Init is called another time and waits for another input.

After scanning the input form the playing using the scanner method a it starts checking if the input is within the bounds of the board and if not the Init called once again waiting for the right input to be entered.

Finally, after bypassing these checks the input is inserted in the board and the CheckWin(), CheckWinner(), IsBoardfull() methods are being called.

* CheckWinner()

Check either any of the two players have won or the game has ended as draw by calling the function checkWin() ,if the return==1 then there are two options, either player1 or player 2 has won ,

After that the function compares the last played symbol(X||O) with player 1 symbol(X) and player 2 symbol(O) to finally judge which player has won.

If the checkWin() returned 0 and IsBoardFull()returned 1 the program detects that the game has ended draw and it’s a tie.

* CheckWin()

For checking whether any win or lose condition is reached. By applying the tic tac toe game rules:

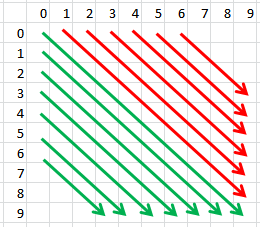
Horizontal check:

By taking the the RowNum chosen by the user as a parameter it starts looping through the columns of that row from column 0 till the length of the array[i](7-1). While looping, the function keeps comparing every position with the current player symbol if matched the counter increments until reaching the winning situation three in a row and if an opponent symbol was found the counter is set to be zero again.

Vertical check:

By taking the the ColNum chosen by the user as a parameter it starts looping through the rows of that column from row 0 till the length of the array (6-1). While looping, the function keeps comparing every position with the current player symbol if matched the counter increments until reaching the winning situation three in a column and if an opponent symbol was found the counter is set to be zero again.

Diagonals checking:

This check is done in two functions by checking the upper triangle of the matrix and the lower one in both directions as it starts from the first row to the third one (As 6 rows) and then do the same for the columns to the end of it (7 in our case), all the previous is done once more for the lower triangle in both directions is illustrated in the photo.

* IsBoardFull()

Check whether the board got completely full or not by looping through the whole grid using a 2 for loops and keep on checking each position if its occupied with an (-), if a false returned then there is no free positions available and all of the board has been occupied by X,O.

**CODE:**

**package** ttt;

**import** java.util.Scanner;

**public** **class** GameSession { //this class to create the game statues

**private** Player currentplayer;// this is the player who plays now

**private** Board board;//object from class board

**private** Player player1;//object from class player to represent first player

**private** Player player2;//object from class player to represent second player

**private** **boolean** EndGame=**false**;// game current state

**public** GameSession(Board board, Player p1, Player p2) {// constructor

**this**.board = board;

**this**.player1 = p1;

**this**.player2 = p2;

**this**.currentplayer = p1;

}

**public** **void** init() {

Scanner numbers = **new** Scanner(System.***in***);

**while**(**true**) {

//check whether the input is valid or not and sends the valid input to be stored in the array

board.displayBoard();

System.***out***.print(currentplayer.getSymbol() + " Enter a row number (1,2,3,4,5,6): ");

**int** row = numbers.nextInt();

System.***out***.print("Enter a column number (1,2,3,4,5,6,7): ");

**int** col = numbers.nextInt();

**if**(row-1 < 0 || col-1 < 0 || row > board.getROWS() || col > board.getCOLUMNS()) {

System.***out***.println("This position is off the bounds of the board! Try again.");

**this**.init();

}

//Check if the position on the board the user entered is empty (has a -) or not

**else** **if**(board.getBoard()[row-1][col-1] != '-') {

System.***out***.println("Someone has already made a move at this position! Try again.");

**this**.init();

}

**else** {

board.putSymbolXorO(row, col, currentplayer.getSymbol());

}

checkWinner(row,col);

**if**(**this**.EndGame==**true**) **break**;

**if**(**this**.currentplayer.equals(player1)) {

**this**.currentplayer=player2;

}

**else** {

**this**.currentplayer=player1;

}

}

}

**public** **void** checkWinner(**int** row,**int** col) {

//Check to see if either player has won

**if**((checkWin(row-1,col-1) == 1) && currentplayer.getSymbol()==player1.getSymbol()) {

System.***out***.println("Player 1 has won!");

board.displayBoard();

EndGame = **true**;

} **else** **if**((checkWin(row-1,col-1) == 1) && currentplayer.getSymbol()==player2.getSymbol()) {

System.***out***.println("Player 2 has won!");

board.displayBoard();

EndGame = **true**;

} **else** {

//If neither player has won, check to see if there has been a tie (if the board is full)

**if**(isBoardFull()) {

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

EndGame = **true**;

}

}

}

//Make a function to check if all of the positions on the board have been filled

**public** **boolean** isBoardFull() {

**for**(**int** i = 0; i < board.getROWS(); i++) {

**for**(**int** j = 0; j < board.getCOLUMNS(); j++) {

**if**(board.getBoard()[i][j] == '-') {

**return** **false**;

}

}

}

**return** **true**;

}

**public** **int** checkWin(**int** rowNum,**int** colNum)

{

// For checking whether any win or lose condition is reached. Returns 1 if win or lose is reached. else returns 0

// colNum is the column number where the last symbol was placed

// rowNum is the row number where the last symbol was placed

**int** count=0;

// Horizontal check

**for** (**int** i=0;i<board.getCOLUMNS();i++)

{

**if** (**this**.board.getBoard()[rowNum][i]==currentplayer.getSymbol())

count++;

**else**

count=0;

**if** (count>=3)

**return** 1;

}

//Vertical check

count=0;

**for** (**int** i=0;i<board.getROWS();i++)

{

**if** (**this**.board.getBoard()[i][colNum]==currentplayer.getSymbol())

count++;

**else**

count=0;

**if** (count>=3)

**return** 1;

}

count=0;

// top-left to bottom-right

**for**( **int** rowStart = 0; rowStart < 3; rowStart++){

count = 0;

**int** row, col;

**for**( row = rowStart, col = 0; row < board.getROWS() && col < board.getCOLUMNS(); row++, col++ ){

**if**(**this**.board.getBoard()[row][col] == currentplayer.getSymbol()){

count++;

**if**(count >=3) **return** 1;

}

**else** {

count = 0;

}

}

}

count=0;

// bottom-right to top-left

**for**( **int** colStart = 1; colStart < 4; colStart++){

count = 0;

**int** row, col;

**for**( row = 0, col = colStart; row < board.getROWS() && col < board.getCOLUMNS(); row++, col++ ){

**if**(**this**.board.getBoard()[row][col] == currentplayer.getSymbol()){

count++;

**if**(count >=3) **return** 1;

}

**else** {

count = 0;

}

}

}

**return** 0;

}

}

**2)** Game

It is the main class to test and play the game. It just creates objects: player1 with symbol X, player2 with symbol O, board, game session and call the function init() to start playing as illustrated before.

**CODE**

**package** ttt;

**public** **class** Game {

// main class to test and play the game.

// it just creates objects and call the function init to start play

**public** **static** **void** main(String[] args) {

Board board = **new** Board();

Player playerOne=**new** Player('X');

Player playerTwo=**new** Player('O');

GameSession gamesession=**new** GameSession(board,playerOne,playerTwo);

gamesession.init();

}

}

**3)** Player

This class just creates an instance of player with one attribute called symbol that carry either X or O for a player and it contains the normal methods which are the getters and setters and the constructor.

The constructor needs the value X or O to create an object from this class.

**Code:**

**package** ttt;

**public** **class** Player {//this class to create players with a specific symbol either X or O

**private** **char** symbol;

**public** Player(**char** symbol) {// constructor

**this**.symbol=symbol;

}

// Getters and Setters

**public** **char** getSymbol() {

**return** symbol;

}

**public** **void** setSymbol(**char** symbol) {

**this**.symbol = symbol;

};

}

**4)** Board

Class board contains 3 attributes which are ROWS (size of rows), Columns (size of columns) and an array of size[ROWS][COLUMNS] to carry the game in.

Methods used:

* The constructor

it initialize the board with dashes in each element (-) and

- its normal getters and setters for all the private instances there.

* displayBoard()

This function is used to show the board each time an entry is done by the user, so it is called each time a play is done by either players, it also displayes the word Board: before the board each time.

* putSymbolXorO()

This function is used to store the value entered by the user either X or O in the place entered by the user, so updates the board each time there is a valid input done by user.

**Code:**

**package** ttt;

**public** **class** Board {

// Create a 6x7 array that represents the board

**private** **final** **int** ROWS = 6;

**private** **final** **int** COLUMNS = 7;

**private** **char**[][] board = **new** **char**[ROWS][COLUMNS];

// -----------------------------------------------------

// Initialize the board with dashes(-)(as empty positions)

**public** Board() {

**for** (**int** i = 0; i < ROWS; i++) {

**for** (**int** j = 0; j < COLUMNS; j++) {

board[i][j] = '-';

}

}

}

// Getters and Setters

**public** **char**[][] getBoard() {

**return** board;

}

**public** **void** setBoard(**char**[][] board) {

**this**.board = board;

}

**public** **int** getROWS() {

**return** ROWS;

}

**public** **int** getCOLUMNS() {

**return** COLUMNS;

}

// Make a function to draw the tic tac toe board (also after each modification)

**public** **void** displayBoard() {

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

**for** (**int** i = 0; i < ROWS; i++) {

// The inner for loop prints out each row of the board

**for** (**int** j = 0; j < COLUMNS; j++) {

System.***out***.print(board[i][j]);

}

// This print statement makes a new line so that each row is on a separate line

System.***out***.println();

}

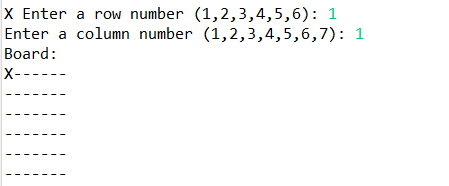
}

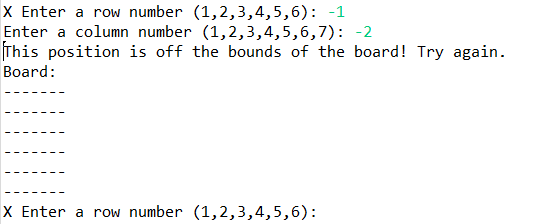
// storing the symbol in the 2D Array.

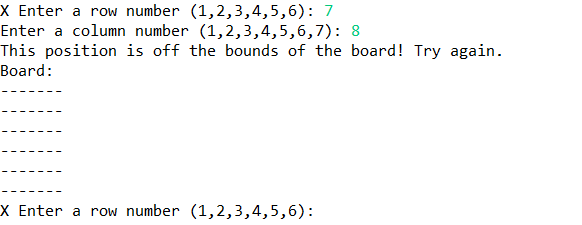
**public** **void** putSymbolXorO(**int** row, **int** column, **char** symbol) {

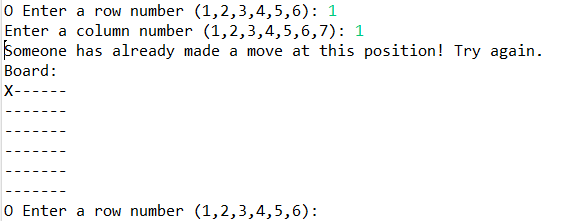
**this**.board[row - 1][column - 1] = symbol;}}

**USER MANUAL AND SAMPLE RUNS**

**When you run the game you are prompted to chose a row and a column number for placing the symbol at.**

**If the chosen co-ordinates are out of the bounds you will be asked to re-enter a new valid co-ordinates**

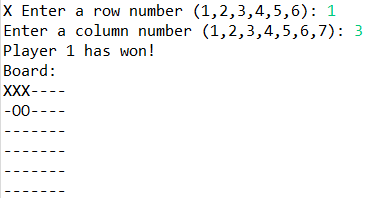
****

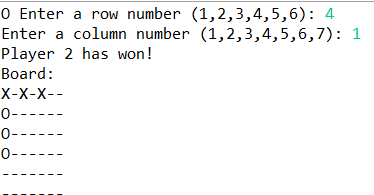
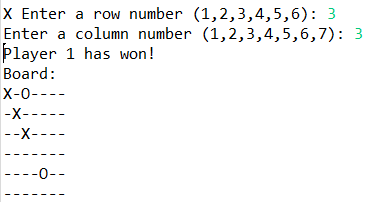
**Also if the position chosen is occupied with an opponent symbol you are not allowed to over-write it and have to chose another slot for playing**

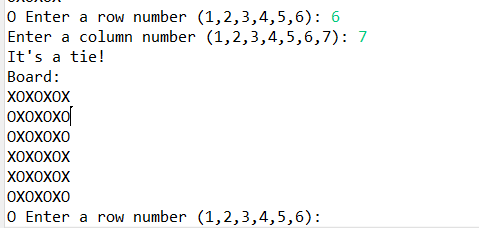
* ***Winning rules:***

**For winning the game each player should target to get three consecutives of his symbol without letting the opponent player to break his streak by his symbol. The three consecutives could be in any order over the grid:**

* **Three in a row:**

****

* **Three in a column:**
* **Three across the diagonals**
* ***Draw rules:***

**If the board got fully occupied by both symbols without getting any three consecutives, then it’s a tie and the game is draw.**