NIRMA UNIVERSITY
INSTITUTE OF TECHNOLOGY
B.Tech in EC/IC Technology - Semester V
2CSOE76 – Object Oriented Programming
Open Elective Subject [CSE Department]
Comprehensive Evaluation – October 2020

Name: Ambika Lakhera **Roll No. : 18BEC006**

Problem Statement - Design a Java Model for a Tic Tac Toe Game. In this game there are two players, each assigned a symbol "X" and "O". The game is played in a 3*3 structure. Each player keeps their symbol token one by one in a single position of 3*3 structure. When a particular player's 3 symbol token aligns horizontally, vertically, and diagonally, the player wins. Model this game using Classes, Functions and Arrays. One can show the intermediate result of the cell occupied using Array. Signify which OOP concepts are used and how they are important.

Steps:

1. **Create a class for players with a name and symbol associated.**

```
class Player
{
        String name;
        char symbol;

        Player(String name, char symbol)
        {
                this.name = name;
                this.symbol = symbol;
        }
}
```

Concept used: classes, use of this

2. **Create a class called Table to store a 3x3 array of characters**

```
class Table
{
        char array[][] = new char[3][3];
        Player currentPlayer;

        Table()
        {
                for(int i = 0; i<3; i++)
                {
                        for(int j = 0; j<3; j++)
                        {
                                array[i][j] =  ' ';
                        }
                }
        }
```

Initialize it with some character as initially all the characters will be initialized with NULL character and to check if any position in the array is occupied by some character, we need to compare it with some character that represents an empty space. The Table class also has a reference of an object of Type Player which will store the information of the current player. This is needed to display the name of the player who wins the game and ends the program.

Concepts used: multidimensional arrays

3. **Create objects of the players and the TicTacToe Table and introduce a method to show the status of the game i.e., the 3x3 structure**

```
void show_Table()
{
        for(char []a: array)
        {
                for(char b : a)
                {
                        System.out.print("|" + b + "|");
                }

                System.out.println();
        }
        System.out.println();
        System.out.println();
}
```

Concept used: For-each loop, method in classes

```
class TicTacToe
{
        public static void main(String args[])
        {
                Player P1 = new Player("P1",'X');
                Player P2 = new Player("P2",'O');

                Table ttt = new Table();
                ttt.show_Table();
```

4. **Take input of the players turn by turn and if the inputs are valid positions and vacant**

```
class TicTacToe
{
        public static void main(String args[])
        {
                Player P1 = new Player("P1",'X');
                Player P2 = new Player("P2",'O');

                Table ttt = new Table();
                ttt.show_Table();

                while(1>0)
                {

                        ttt.gameON(P1);
                        ttt.show_Table();
                        if(ttt.anyone_win() || ttt.tableFilled())
                                break;

                        ttt.gameON(P2);
                        ttt.show_Table();
                        if(ttt.anyone_win() || ttt.tableFilled())
                                break;
                }
        }
}
```

Create a method called gameON() in the class Table and to make this program run continuously, keep taking the inputs sequentially inside a while loop which breaks only when any player wins the game or when the table gets filled.

Concept used: while loop , if , break

```
void gameON(Player p)
{
        currentPlayer = p;
        Scanner take_ip = new Scanner(System.in);
        int x,y;

        try
        {
                do
                {
                        System.out.println("Player: " + p.name + " Symbol: " + p.symbol);
                        System.out.print("Enter the position in the table: ");

                        x = take_ip.nextInt();
                        y = take_ip.nextInt();

                } while (notVacant(x-1,y-1));

                array[x-1][y-1] = p.symbol;
        }



        catch(Throwable exc)
        {
                System.out.println("Enter a valid position within (1,1) to (3,3)");
        }


}
```

The gameON() method takes the object Player p as an input argument and asks for the input of the position the player with its associated symbol wants to enter in the 3x3 structure. This is done using a do while loop which executes once but keeps repeating itself until and unless the position at which the player wants to enter its associated symbol is vacant. This is done with the help of the method notVacant(int x, int y) which takes the position coordinates as inputs and returns the boolean value if the position at the table is empty or not.

```
boolean notVacant(int x, int y)
{
        boolean value = false;

        if(array[x][y] != ' ')
        {
                value = true;
                System.out.println("Not vacant, please try for some different place");
        }

        return value;
}
```

The if condition inside the method compares the array element at position [x][y] to ‘ ‘ which was the character with which the 3x3 array was initialized.

If it is not ‘ ‘, then it returns true, i.e., the while loop continues and keeps taking the input of the user until it enters the position which is vacant.

If the user enters any invalid position that is out of the limit of the array[3][3], or if it enters any non-integer value, an exception is raised which is caught and handled by the Throwable class.

Concept used: Exception Handling , Catching Subclass exception

5. **Check if any player has won the game or if the table is all filled after taking the inputs of a player , if it is true, then end the program by breaking the while loop**

```
class TicTacToe
{
        public static void main(String args[])
        {
                Player P1 = new Player("P1",'X');
                Player P2 = new Player("P2",'O');

                Table ttt = new Table();
                ttt.show_Table();

                while(1>0)
                {

                        ttt.gameON(P1);
                        ttt.show_Table();
                        if(ttt.anyone_win() || ttt.tableFilled())
                                break;

                        ttt.gameON(P2);
                        ttt.show_Table();
                        if(ttt.anyone_win() || ttt.tableFilled())
                                break;
                }
        }
}
```

The condition for checking if the game is won or not first written and is evaluated first than checking if the table is filled. This is because it may happen that after

entering the symbol of the current player, the player may win the game and also the table may get filled. But we don't want to check the condition of the filled table if the game ends by winning.

Concept used : Logical OR

```
boolean tableFilled()
{
        boolean filled = true;

        for(int x = 0; x<3; x++)
        {
                for(int y = 0; y<3; y++)
                {
                        if(array[x][y] == ' ')
                        {
                                filled = false;
                                break;
                        }
                }
        }

        if(filled)
                System.out.println("Table is filled! It's a draw");

        return filled;
}
```

Run the for loop and check wherever any vacant space is present, i.e., if any character of the 3x3 structure is ‘ ‘, then break the loop and return false, else print the message that it’s a draw .

```
boolean anyone_win()
{
        boolean answer = false;   // NO ONE WON
        testTable test = new testTable(array);   // PASS TABLE AS AN OBJECT TO TEST IT

        if(test.diagonal1() || test.diagonal2() || test.horizontal() || test.vertical() )
        {
                answer = true;
        }
        if(answer == true)           // CURRENT PLAYER HAS WON THE GAME
        {
                System.out.println(currentPlayer.name + " with symbol " + currentPlayer.symbol + " wins the game");
        }
        return answer;
}
```

Check if any 3 symbols are aligned in the two diagonals, or horizontally or vertically. If it is true, then return true implying that the game is won by the current player else return the default value false. This is done by passing the character array to check an object test of type testTable which has all the methods checking the testing conditions for which the player wins the game.

6. **Check if 3 symbols are aligned diagonally, vertically or horizontally everytime a player enters his symbol - for this, create a class testTable and pass the 3x3 array structure as an argument to the object of the type testTable**

```
class testTable
{
        char array[][] = new char[3][3];

        testTable (char[][] array)
        {
                this.array = array;
        }
```

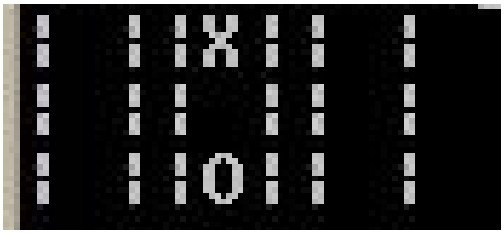
The testTable has a constructor which stores the passed character array to its data member - a character array of 3x3.

```
boolean diagonal1()
{
        boolean diagonally_yes = true;

        char current = array[0][0];
        char previous = array[0][0];

        for(int x = 0; x<3; x++)
        {
                current = array[x][x];
                if( ((previous == ' ') && (current == ' ')) || (previous != current))
                {
                        diagonally_yes = false;
                        break;
                }
                previous = current;
        }

                return diagonally_yes;
}
```

The testTable has a method called diagonal1() which returns the boolean value if 3 symbols of a particular character are aligned along one diagonal starting from the top left to bottom right. For this, run a for loop which breaks if the previous encountered character is not the same as the current encountered character. The for loop runs only for the diagonal elements. Hence, if the loop breaks, then there is no alignment and the method should return false.

As the loop begins from the top left element, the previous character is initialized to the top left element by default.

Also, there is a possibility that the table is partially filled and the diagonal elements are not occupied but yet the method returns true. This is because the table was initialized by ‘ ‘ and here, all the elements in the for loop would be the same hence, the loop may not break. To avoid this, we must check if the previous and the current character are not ‘ ‘ and hence must not break the loop in that case.



Concept used: Logical AND

```
boolean diagonal2()
{
        boolean diagonally_yes = true;

        char current = array[0][2];
        char previous = array[0][2];

        for(int x = 0; x<3; x++)
        {
                current = array[x][2-x];
                if( ((previous == ' ') && (current == ' ')) || (previous != current))
                {
                        diagonally_yes = false;
                        break;
                }
                previous = current;
        }


        return diagonally_yes;
}
```

Logic for checking the alignment of the symbols in case of second diagonal (bottom left to top right) is similar to the logic for checking the first diagonal. The for loop begins with the element at the bottom right and ends at the top left. Hence, the previous character is assigned the bottom right character value.

```
boolean horizontal()
{

        boolean horizontally_yes = true;
        boolean end_of_row;

        char current;
        char previous;

        for(int x = 0; x<3; x++)
        {
                current = array[x][0];
                previous = array[x][0];
                end_of_row = true;

                for(int y = 0; y<3; y++)
                {
                        current = array[x][y];
                        if(((previous == ' ') && (current == ' ')) || (previous != current))
                        {
                                end_of_row = false;
                                horizontally_yes = false;
                                break;
                        }
                        previous = current;
                }

                if(end_of_row == true)
                {
                        horizontally_yes = true;
                        break;
                }
        }


        return horizontally_yes;
}
```

To check the alignment of the symbols horizontally, two for loops are run. Also, every row is checked from left to right. Before any row is checked, the current and the previous values are assigned the value of the left most character. If the current and the previous values are not the same, then the loop is broken and the end of row has not been reached. Next row is checked for the alignment of the symbols. If it happens that the inner loop is not broken, that implies that alignment is done and the current player has won the game. Hence, we must not check other rows. Thus, the end_of_row has its value true preserved which was assigned to it at the beginning of the outer for loop. If after executing the inner loop, the end_of_row is still true, then we get out of the outer for loop and return the true value that horizontally, an alignment is found.

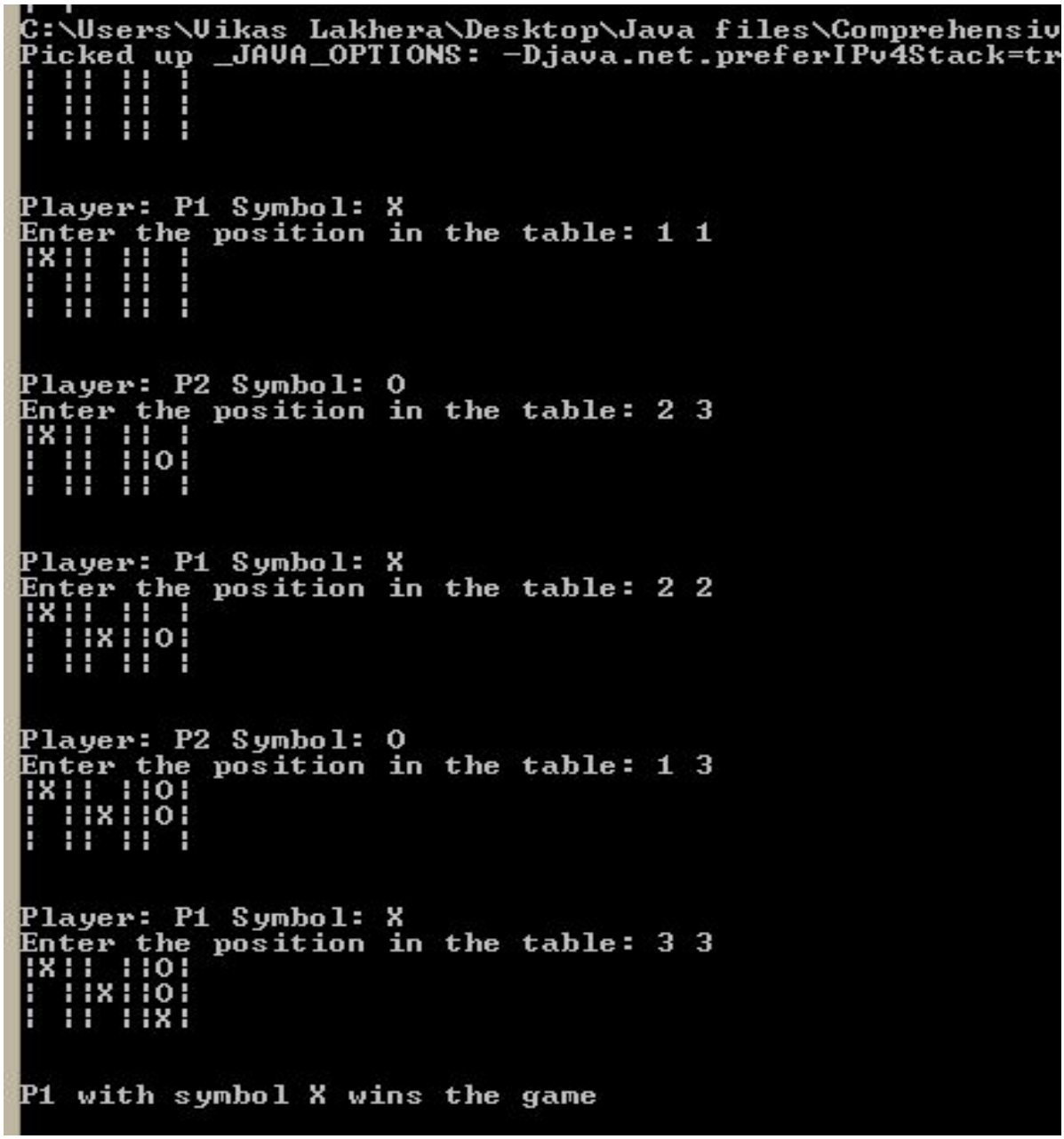
```
boolean vertical()
{

        boolean vertically_yes = true;
        boolean end_of_column = true;

        char current;
        char previous;

        for(int x = 0; x<3; x++)
        {
                current = array[0][x];
                previous = array[0][x];
                end_of_column = true;

                for(int y = 0; y<3; y++)
                {
                        current = array[y][x];
                        if(((previous == ' ') && (current == ' ')) || (previous != current))
                        {
                                end_of_column = false;
                                vertically_yes = false;
                                break;
                        }
                        previous = current;
                }

                if(end_of_column == true)
                {
                        vertically_yes = true;
                        break;
                }
        }


        return vertically_yes;
}
```

Similar to the logic written for the horizontal(), logic for vertically() can also be written. Now, the loops are checked for vertical columns.

Concept used: usage of flags

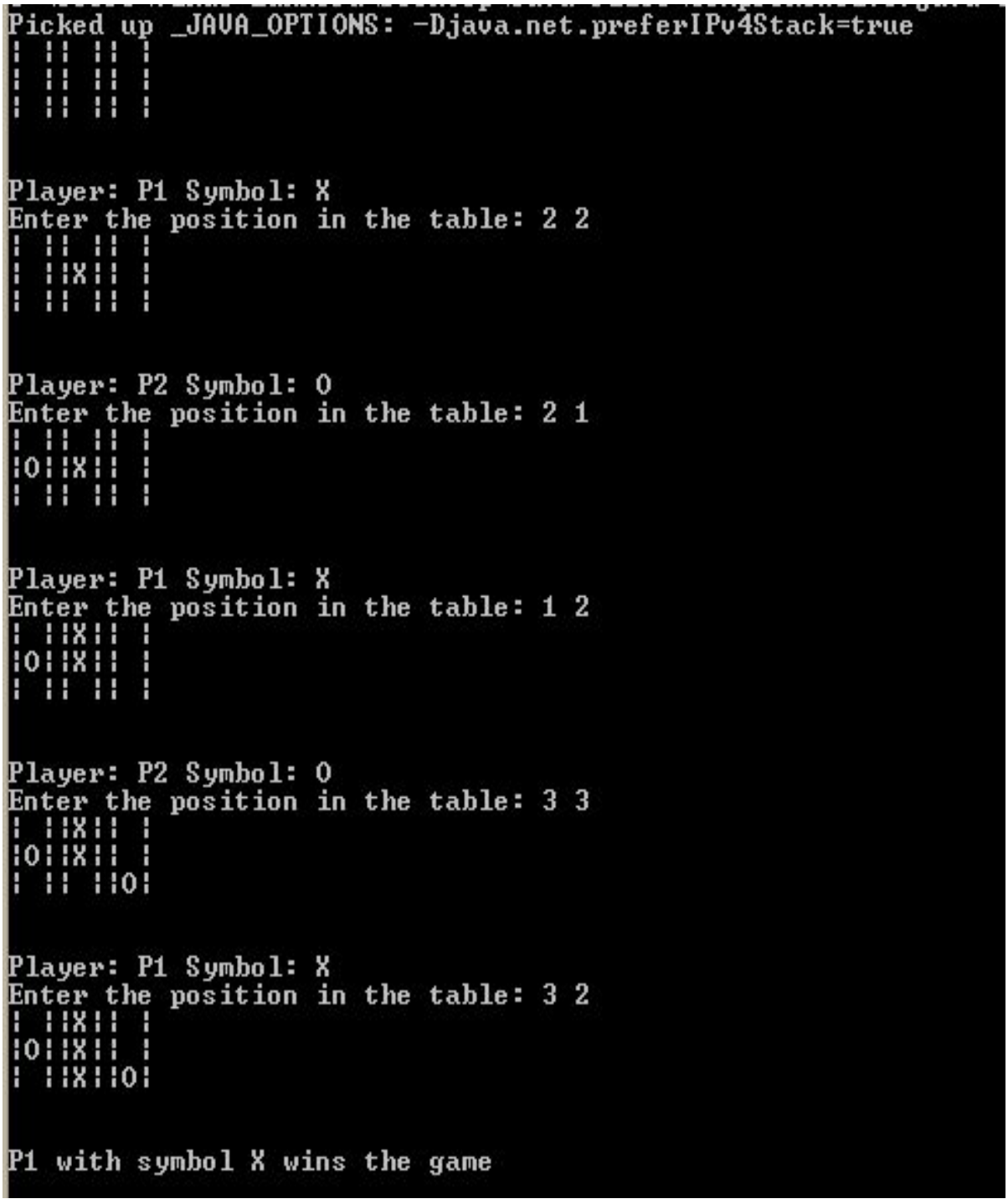
Output:

1. **Aligning of symbols along the Diagonal 1**

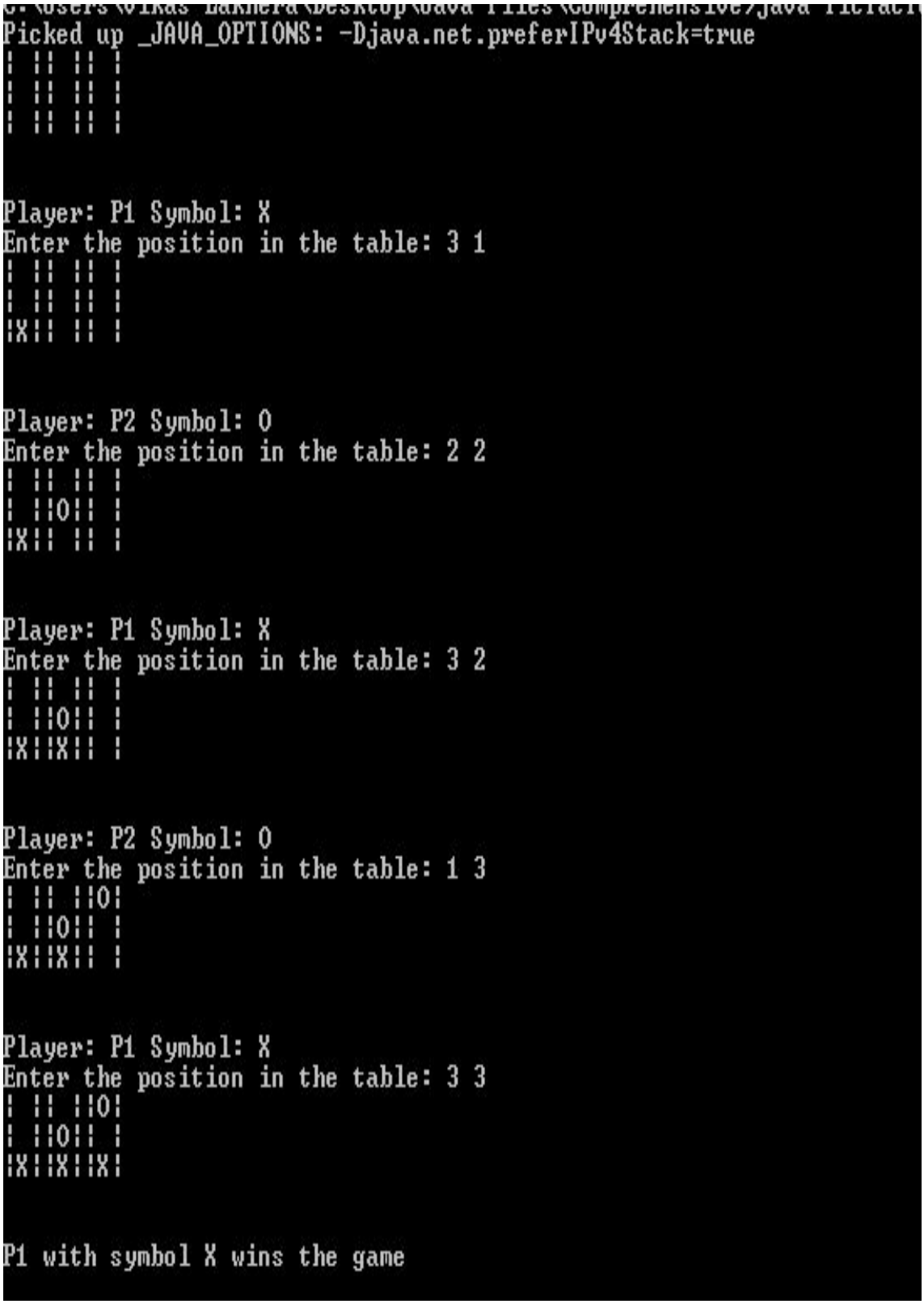


2. **Alignment of symbols along Diagonal 2**

3. **Alignment of symbols vertically**



4. **Alignment of symbols horizontally**

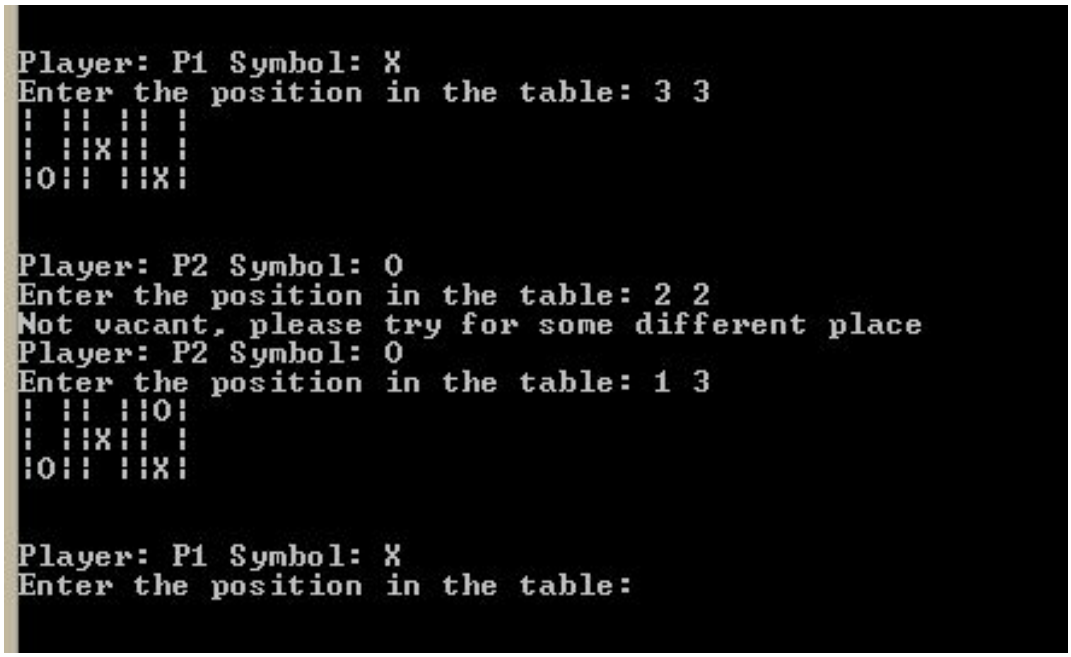


5. Table filled - Draw!

```
Picked up _JAVA_OPTIONS: -Djava.net.preferIPv4Stack=true
|  ||  ||  |
|  ||  ||  |
|  ||  ||  |


Player: P1 Symbol: X
Enter the position in the table: 2 2
|  ||  ||  |
|  ||X||  |
|  ||  ||  |


Player: P2 Symbol: O
Enter the position in the table: 3 1
|  ||  ||  |
|  ||X||  |
|O||  ||  |


Player: P1 Symbol: X
Enter the position in the table: 2 1
|  ||  ||  |
|X||X||  |
|O||  ||  |


Player: P2 Symbol: O
Enter the position in the table: 3 2
|  ||  ||  |
|X||X||  |
|O||O||  |


Player: P1 Symbol: X
Enter the position in the table: 1 2
|  ||X||  |
|X||X||  |
|O||O||  |


Player: P2 Symbol: O
Enter the position in the table: 1
1
|O||X||  |
|X||X||  |
|O||O||  |


Player: P1 Symbol: X
Enter the position in the table: 1 3
|O||X||X|
|X||X||  |
|O||O||  |
```

```
Player: P2 Symbol: O
Enter the position in the table: 2 3
|O||X||X|
|X||X||O|
|O||O||  |


Player: P1 Symbol: X
Enter the position in the table: 3 3
|O||X||X|
|X||X||O|
|O||O||X|


Table is filled! It's a draw
```

6. **Checking vacancy of the position**



7. **Until a valid position is entered, don't update the 3x3 structure table**