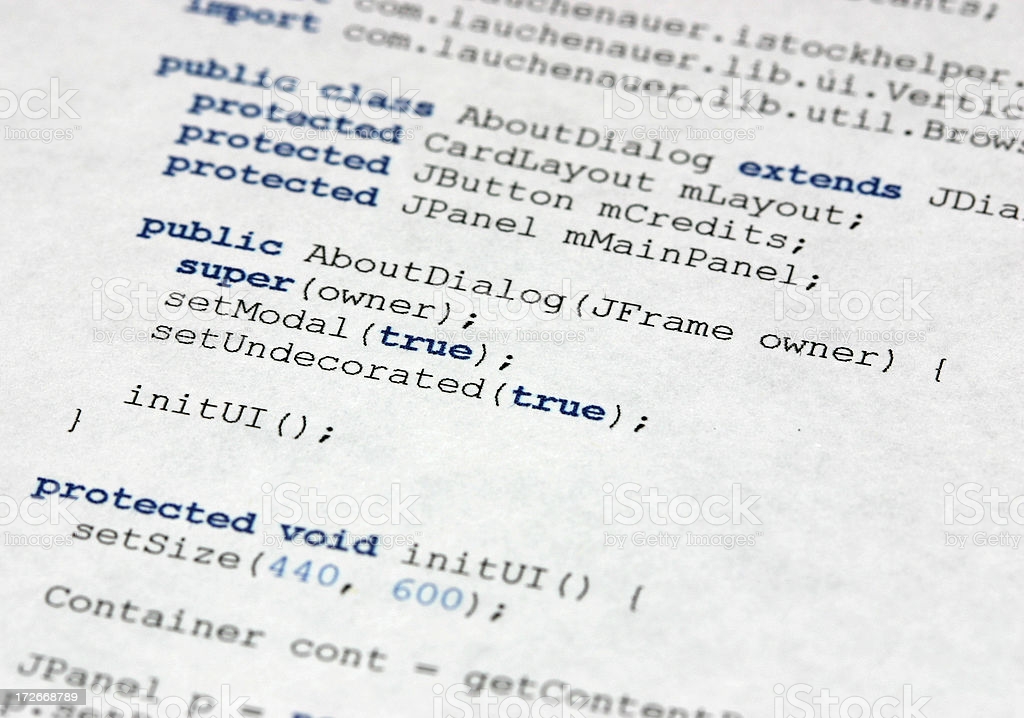
Programming

Assignment 1 report



**First: Description:**

We start the program by making a package named tictactoe and putting a class TicTacToeGame inside it.

Then we used the main method in Java which is

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

First, we start creating 6\*7 array of type char named tictak that represents tic tac toe board where all of positions are dashes “-“ or (empty positions) ;

**char** tictak[][] = **new** **char**[6][7];

Then we define global variables as (name of players, symbols used in playing)

Then defining two variables of type String which represents the names of the two players String x,y;

And then, we start reading inputs (asking for the names) from dialog box by using

x = JOptionPane.*showInputDialog*("Player1, What is your name?");

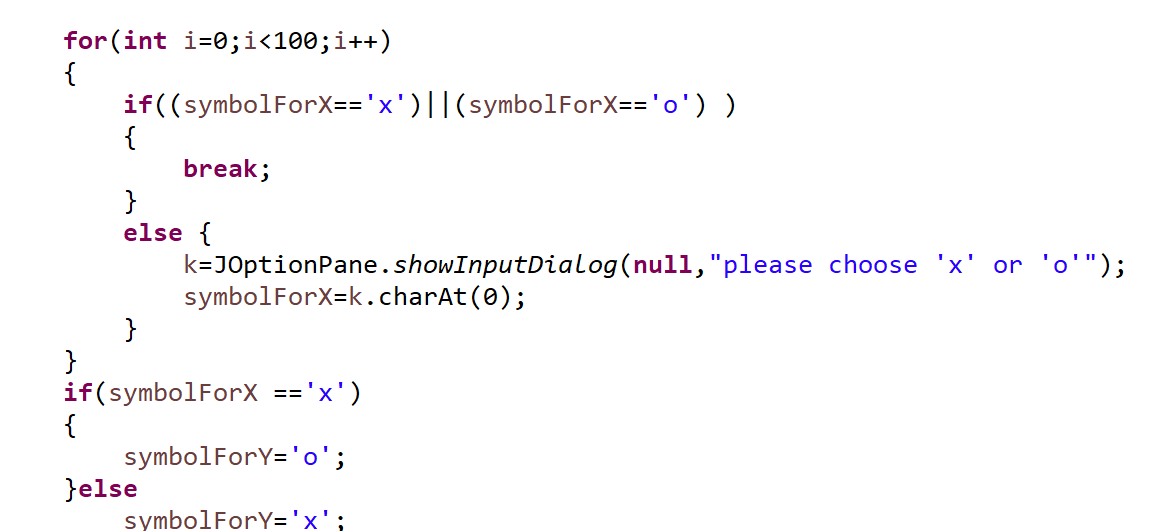
And similarly player2 or (y)

Using the library **import** javax.swing.JOptionPane; as a helping method in the scan or the output and result appears in the dialog box.

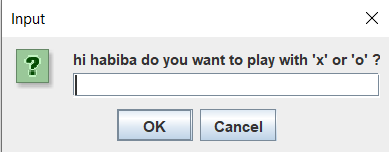
Then we want to make the first player to choose whether to play with ‘x’ or ‘o’ by initializing two variables of type char

**char** symbolForX,symbolForY; where ‘X’ is first player

that if the first player chooses to play with ‘x’, it will break out of the (for loop) and make the symbol for player2(Y) =’o’ and vice versa as shown in **fig 1.1**.

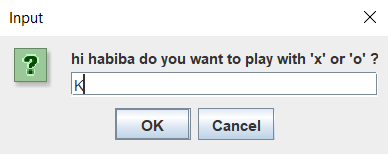


**Fig 1.1**



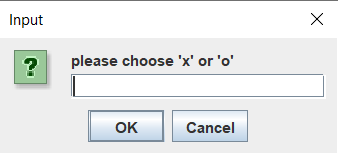
**Fig 1.2**

We check for the validity of scanning in the (for loop) in **fig 1.1** that if the player entered any character else as ‘k’ for example instead of ‘x’ or ‘o’ .

**fig 1.3**

**Fig 1.3**

It will print this message and ask him to re-enter again until the player enters a valid character (‘x’ or ‘o’) as shown in **fig 1.4**



**Fig 1.4**

Create a playerTurn boolean that is true if it is player1's turn and false if it is player2's turn

**boolean** playerTurn = **true**;

Create a endGame Booleanand using the not of it as the condition in the while loop

**boolean** endGame = **false**;

A playAgain Boolean that will be in while loop to ask the players to play again or not after finishing first game.

**boolean** playAgain =**true**;

**while**(!endgame)which means “while the game has not come to an end yet”.

Thus, using the method *drawBoard*(tictak) inside while loop explains the continuity of drawing the board while the game is still going on.

We need to scan the indexes of the rows and columns

but first we initialize rows and cols of type String

String rows,cols;

rows= JOptionPane.*showInputDialog*("Please enter row bet 0 & 5:");

cols= JOptionPane.*showInputDialog*("Please enter col bet 0 & 6:");

And then convert them to integer to enter the inputs

row=Integer.*parseInt*(rows);

col=Integer.*parseInt*(cols);

But first, initializing row and col or any two variables of type ‘int’

**int** row = 0;

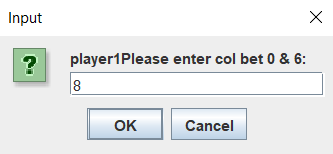
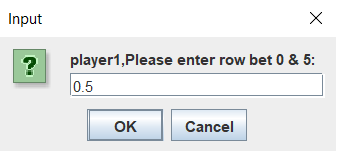
**int** col = 0;

So, as shown in **Fig 1.5,** the player inputs an index for row and column and we check the validity of the numbers and the positions chosen by the players.

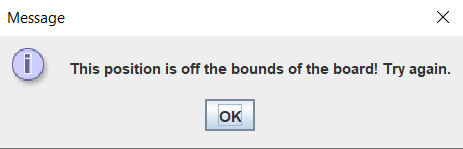


**Fig 1.5**

We make a validity for the scan of rows and columns to check that the user has not entered a number out of the bounds or a decimal number as following;



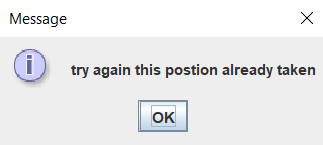
Thus, the message in **Fig 1.6**  appears and will ask the player to re-enter the index of the row and the column and will repeat the message until the player enters valid numbers.



**Fig 1.6**

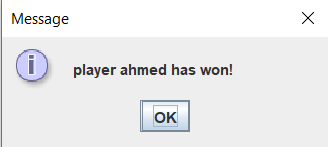
We make a method  **if**(tictak[row][col]!='-') to check if the position the player has chosen ,is not empty.

So, it will print this message in **Fig 1.7** and then will ask the player to re-enter the index until the player enters a valid index.



**Fig 1.7**

After entering a valid input for row and column the program will check if there exist three consecutive rows or columns or diagonals with the same symbol (‘x’ or ‘o’)  
if found it will print the name of the winner as shown in **Fig 1.8**



**Fig 1.8**

Then, we initialize two variables of type “int”

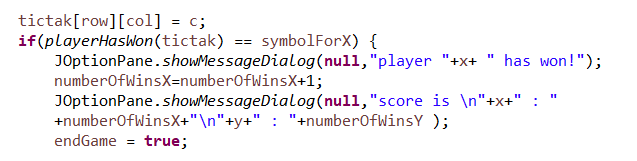
**int** numberOfWinsX=0;

**int** numberOfWinsY=0;

To count the number of times player1 has won for example, we make score between them,

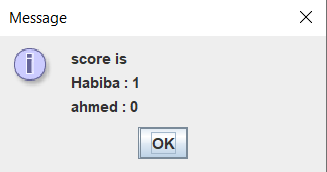
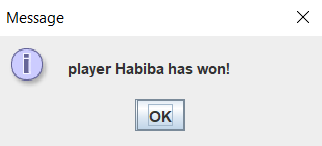
where X is player1 and Y is player2.

We use the code below to enter the winner in the dialog box and for the score to count the number of winning times and similarly for player2 if he wins.

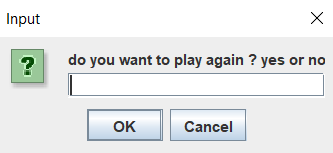


If there are no three consecutive elements it will print the board is full.

The message dialog box in the following shows the winner and make score between the two players.



It also asks them if they want to play again or not as shown in **Fig 1.9** and there is a validity that they must input ‘yes’ or ‘no’ only or an input dialog box will re-ask the player for re-rentering.



**Fig 1.9**

**second: Algorithm:**

We have three methods at our program :

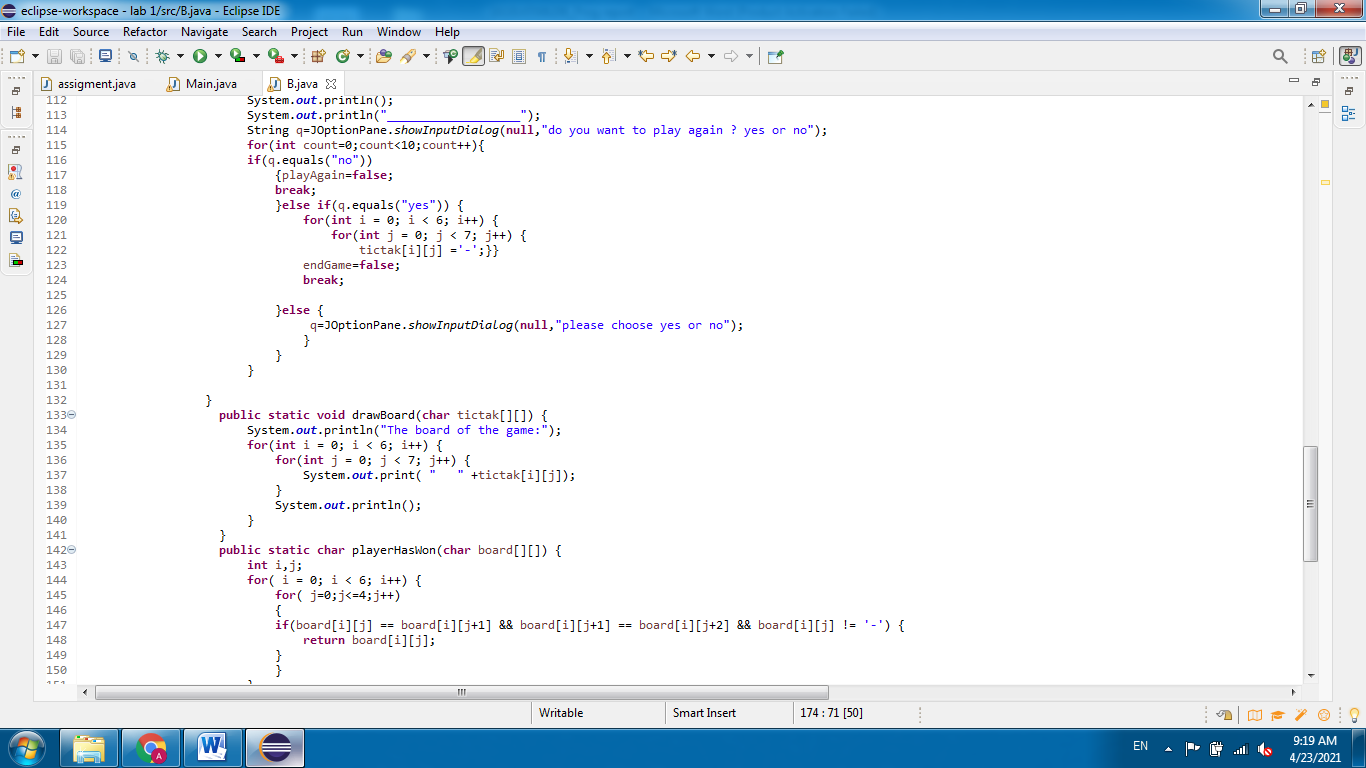
- public static void drawBoard(char tictak[][])

- public static char playerHasWon(char board[][])

- public static boolean boardIsFull(char tictak[][])

First method:

drawBoard method is printing our board using two for loop one for the rows and the other for the columns taking in consider printing the new rows in new line



Second method:

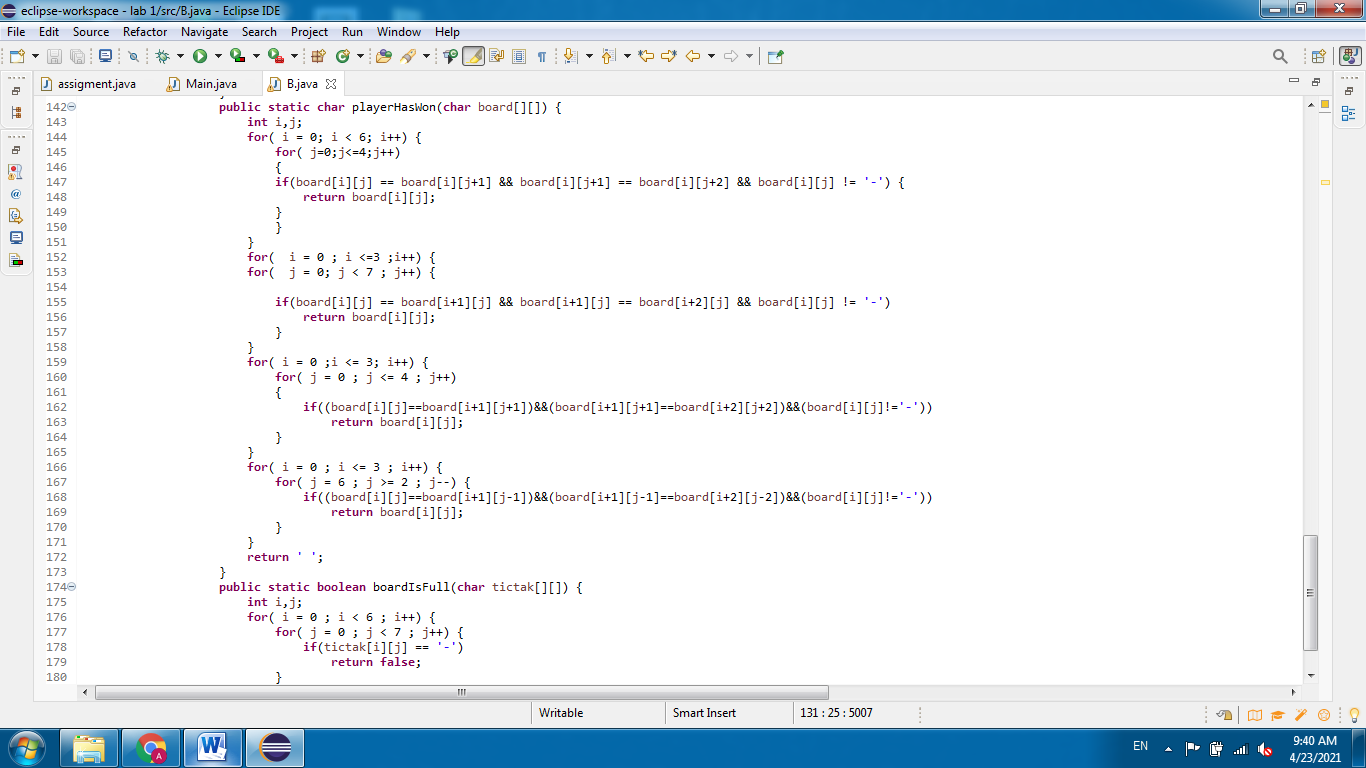
playerHasWon method with return type char as it returns a 2d array of char, we use it to identify the cases at which there is three consecutive same symbol

**Table of the board index to explain the algorithm of winning cases**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0,0 | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 |
| 1,0 | 1,1 | 1,2 | 1,3 | 1,4 | 1,5 | 1,6 |
| 2,0 | 2,1 | 2,2 | 2,3 | 2,4 | 2,5 | 2,6 |
| 3,0 | 3,1 | 3,2 | 3,3 | 3,4 | 3,5 | 3,6 |
| 4,0 | 4,1 | 4,2 | 4,3 | 4,4 | 4,5 | 4,6 |
| 5,0 | 5,1 | 5,2 | 5,3 | 5,4 | 5,5 | 5,6 |

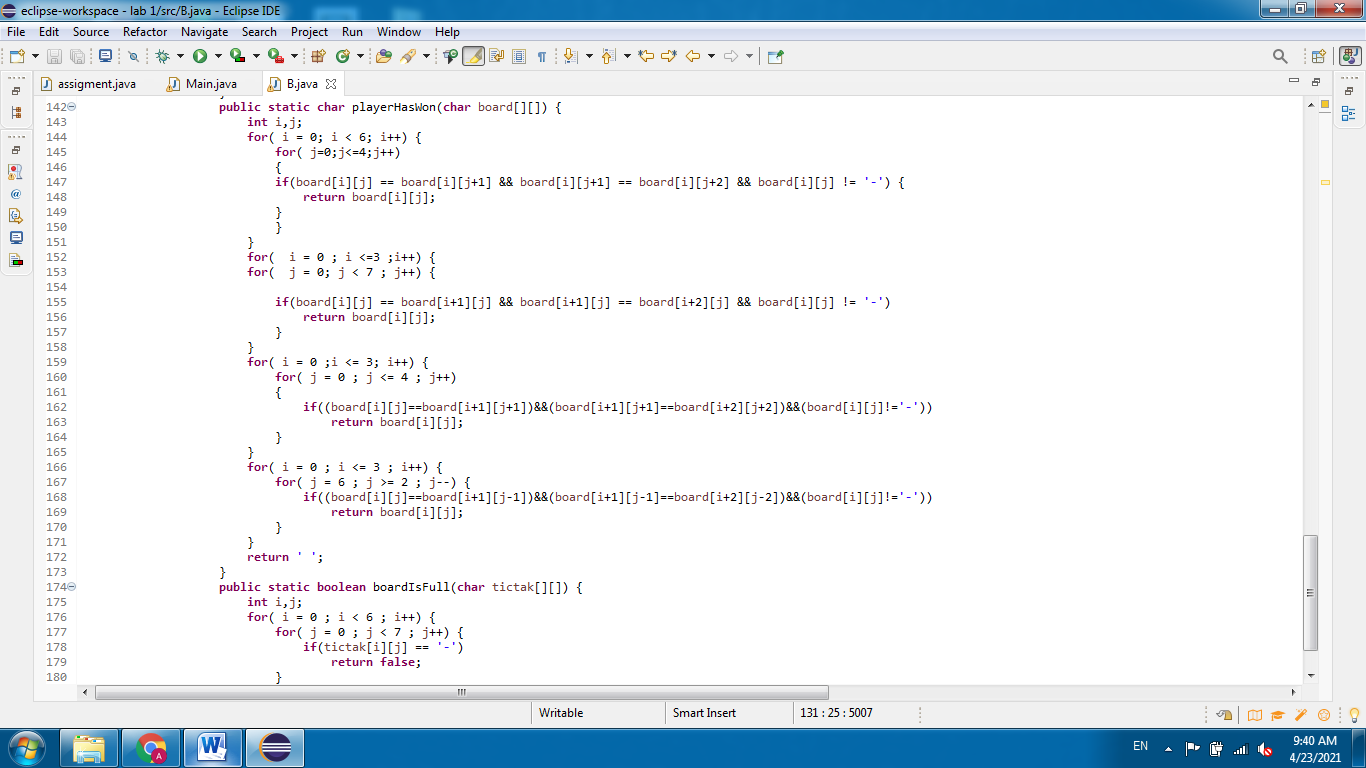
**First**: we are going to cover the row cases, if we look at index we would notice that for the same row the index of the row doesn’t change and index of column increase by one and by two so we made a for loop which check all the rows (i<6) and a for loop to check each element in the same row with the two consecutive elements.

if(board[i][j] == board[i][j+1] && board[i][j+1] == board[i][j+2] && board[i][j] != '-')

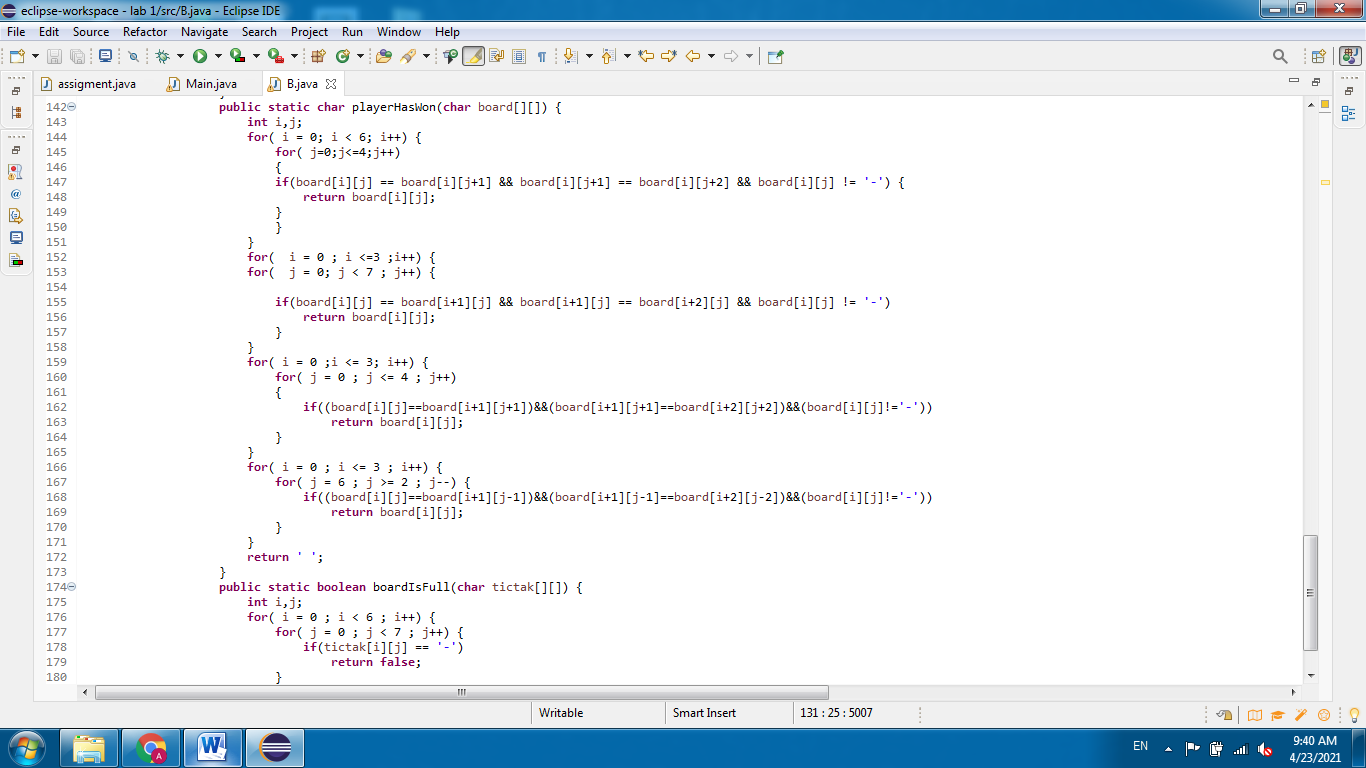


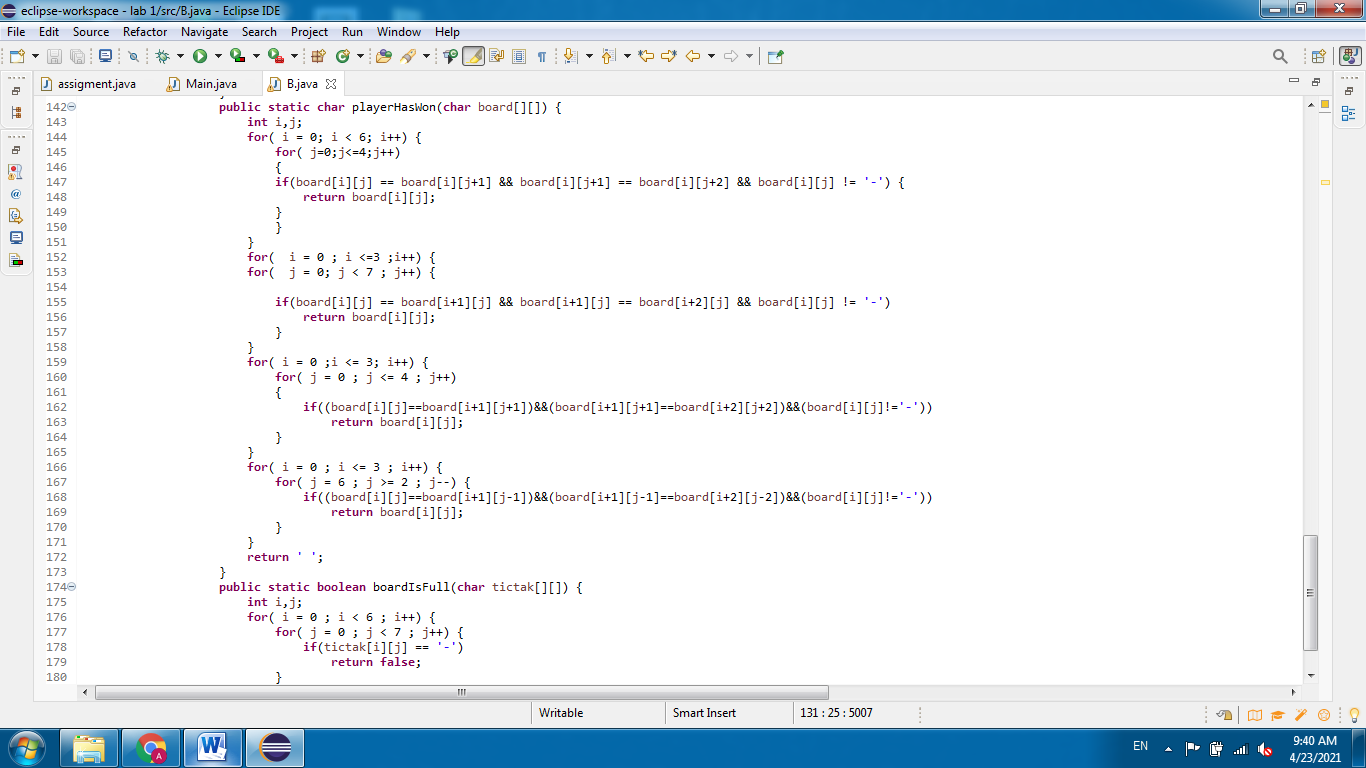
**Note** we made (j<=4) because we don’t need to reach element of 5th column because the reminded columns are only two.

**Second** : Same algorithms are made to check the column cases with reversing the check



**Third**: To cover the cases of diagonal we found that there is two main cases for diagonal each one with different algorithm  
a) For the left diagonal we notice that the index of both columns and rows increase by 1

 b) For the right diagonal we notice that the row index increase by 1 and the column index decrease by 1



Third method:

We use it to check if board is full or not by making it return a Boolean value, false if it contain the char ‘-’ and return true if it doesn’t contain ‘-’

