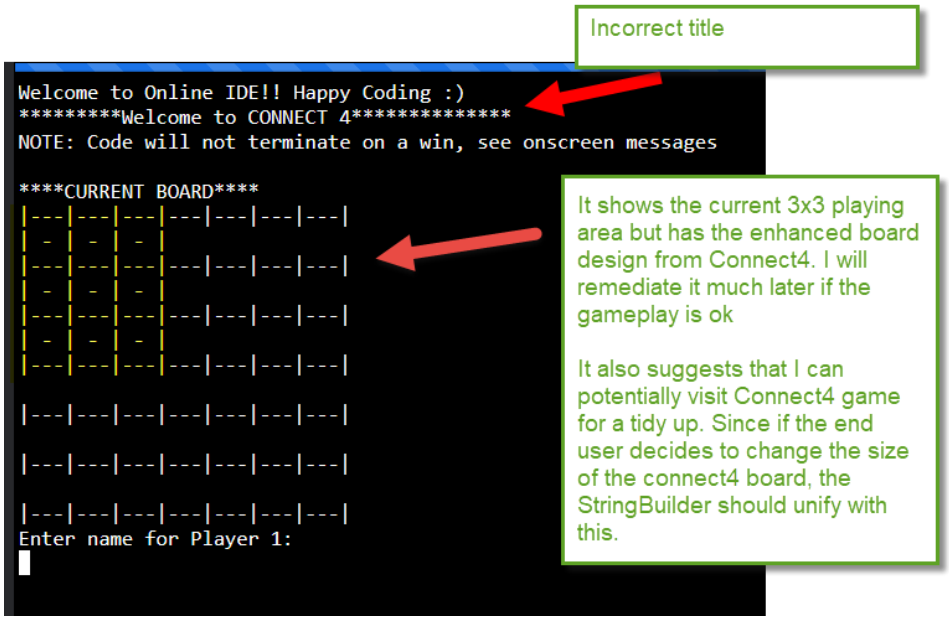
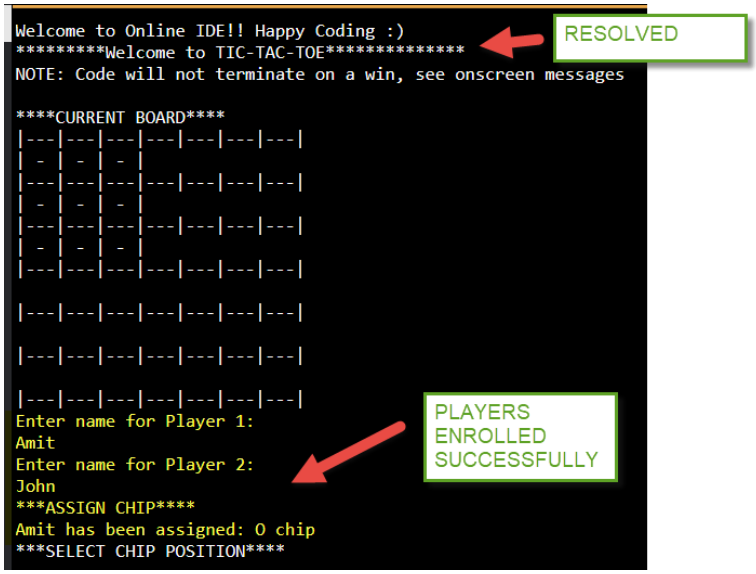
I have now changed as much logic as possible and the code is in an error free state for compilation.

TEST CASE 1: Executing code and observing the interface



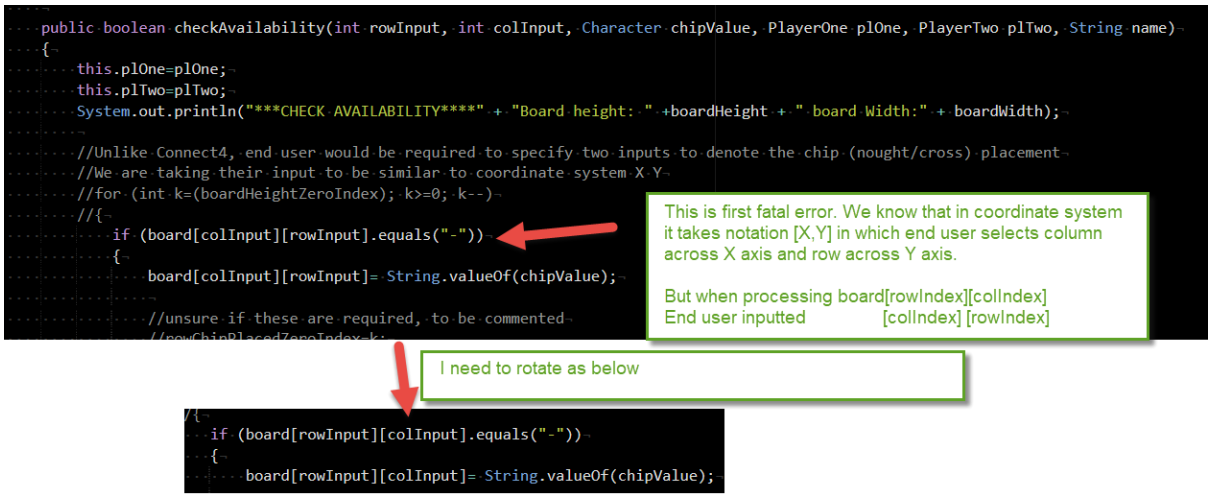
TEST CASE 2: Adjusting name of the game and also performing player selection



TEST CASE 2: Selecting chip position

Based on these errors, I need to decide if there are too many variables declared resulting in issues.

TEST CASE 2a: identifying issues above and remediating



TEST CASE 2b: Running code again

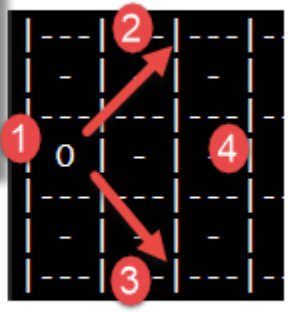
```
Amit(O),  Which column would you like to insert the chip?
1
Amit(O),  Which row would you like to insert the chip?
2
***CHECK AVAILABILITY****Board height: 3 board Width:3
Chip: O will be placed into column: 0 row: 1
Position: [1,2] NOW HAS: O
Last chip inserted O by: Amit(Player 1)

****CURRENT BOARD****
|---|---|---|---|---|---|---|
| - | - | - |
|---|---|---|---|---|---|---|
| O | - | - |
|---|---|---|---|---|---|---|
| - | - | - |
|---|---|---|---|---|---|---|

|---|---|---|---|---|---|---|

|---|---|---|---|---|---|---|

|---|---|---|---|---|---|---|
***CHECK CONNECT FOUR****
```

It has now translated the chip into correct position and also kept the coordinate based system [X,Y] outputted to the end user

I am now ready for the next test phase

TEST CASE 3: Examining the checks undertaken

We can see from this position, performing SE and NE were required on a Connect4 board. But on Tic-Tac-Toe it serves no purpose whatsoever




```
***CHECK CONNECT FOUR****
VERTICAL CHECK => DOWNWARDS*****
HORIZONTAL CHECK => RIGHT*****
HORIZONTAL CHECK => LEFT*****
DIAGONAL CHECK 1
INSIDE D1 - Diagonal north east check
DIAGONAL CHECK 2
DIAGONAL CHECK 3
INSIDE D3 - Diagonal south east check
value of offset: 1
DIAGONAL CHECK 4
***ASSIGN CHIP****
John has been assigned: X chip
***SELECT CHIP POSITION****
```

I would effectively need to apply same logic for all positions numbered above

Good news is that it has not completed the game

TEST CASE 3a: Implementing relevant logic in the diagonal checks as per above

I have had to consider two additional exempt locations for each diagonal move

```
if (columnChipPlacedZeroIndex!=boardWidthZeroIndex && rowChipPlacedZeroIndex!=boardBaseLevel
&& (columnChipPlaced!=2 && rowChipPlaced!=3) && (columnChipPlaced!=1 && rowChipPlaced!=2))
{
    System.out.println("INSIDE D1 - Diagonal north east check");
```

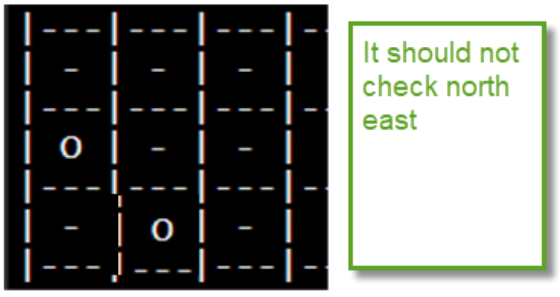
```
if (columnChipPlacedZeroIndex!=0 && rowChipPlacedZeroIndex!=boardHeightZeroIndex
&& (columnChipPlaced!=2 && rowChipPlaced!=1) && (columnChipPlaced!=3 && rowChipPlaced!=2))
{
    System.out.println("INSIDE D2 - Diagonal south west check");
```

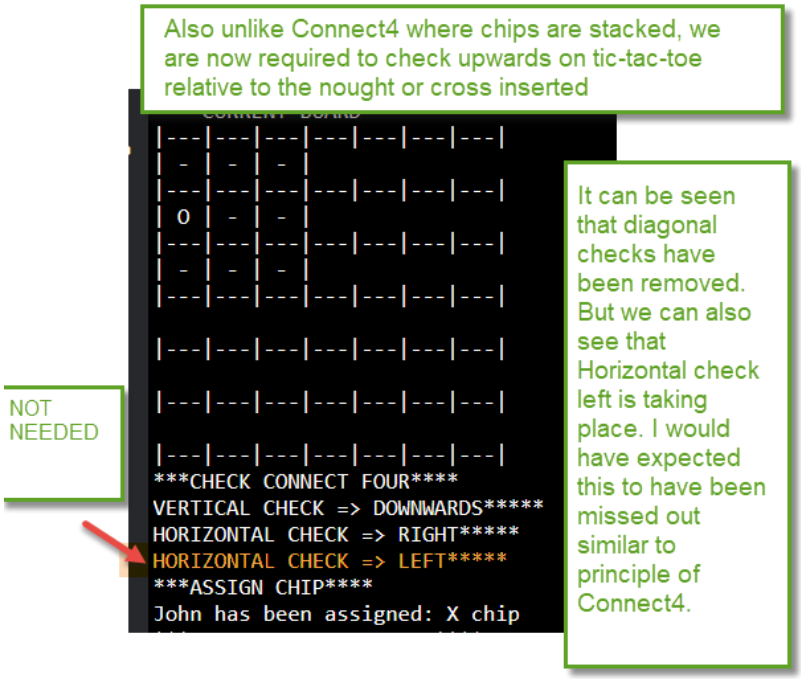
```
if (columnChipPlacedZeroIndex!=boardWidthZeroIndex && rowChipPlacedZeroIndex!=boardHeightZeroIndex
&& (columnChipPlaced!=2 && rowChipPlaced!=1) && (columnChipPlaced!=1 && rowChipPlaced!=2))
{
    System.out.println("INSIDE D3 - Diagonal south east check");
```

```
if (columnChipPlacedZeroIndex!=0 && rowChipPlacedZeroIndex!=0
&& (columnChipPlaced!=3 && rowChipPlaced!=2) && (columnChipPlaced!=2 && rowChipPlaced!=3))
{
    System.out.println("INSIDE D4 - Diagonal north west check");
```

The above could alternatively be kept in an outer if loop, but it seems appropriate maintaining conditions relevant to each scenario easily identifiable.

TEST CASE 3b: Placing the O at all following locations and determining if it bypasses out of bounds diagonals as above

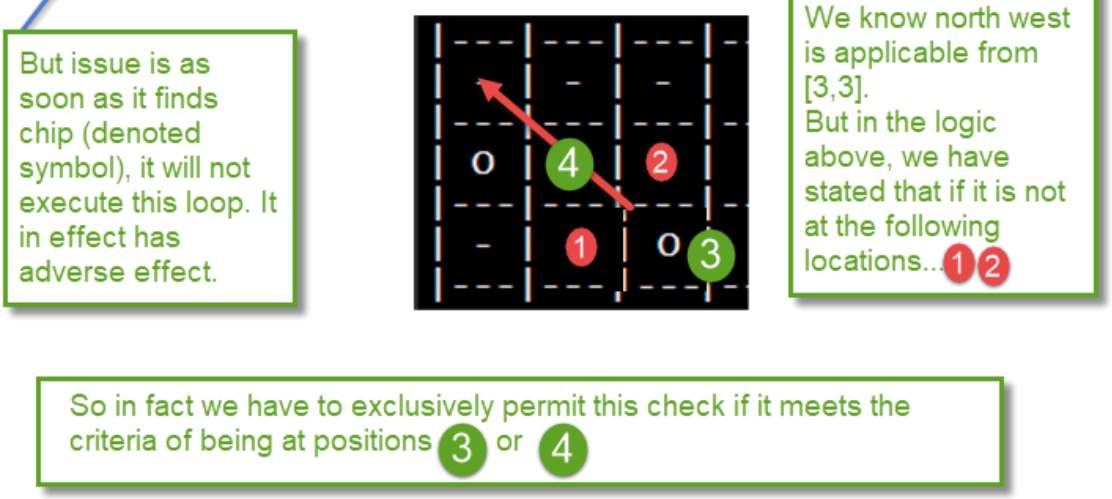




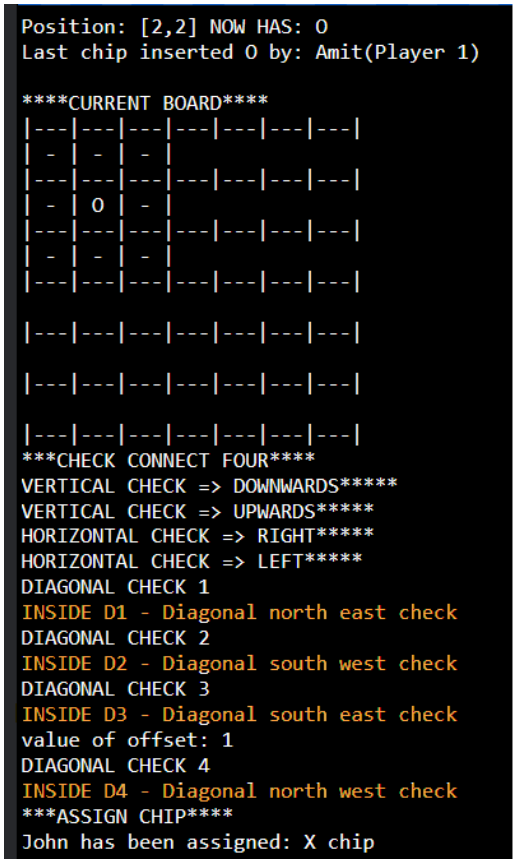
TEST CASE: 3c : resolve all these type issues without documentation
My issues were related to the messages outside of appropriate if statement

Also my test cases in 3a were very detrimental.

```
if (columnChipPlacedZeroIndex!=0 && rowChipPlacedZeroIndex!=0
&& (columnChipPlaced!=3 && rowChipPlaced!=2) && (columnChipPlaced!=2 && rowChipPlaced!=3))
{
    System.out.println("INSIDE D4 - Diagonal north west check");
```

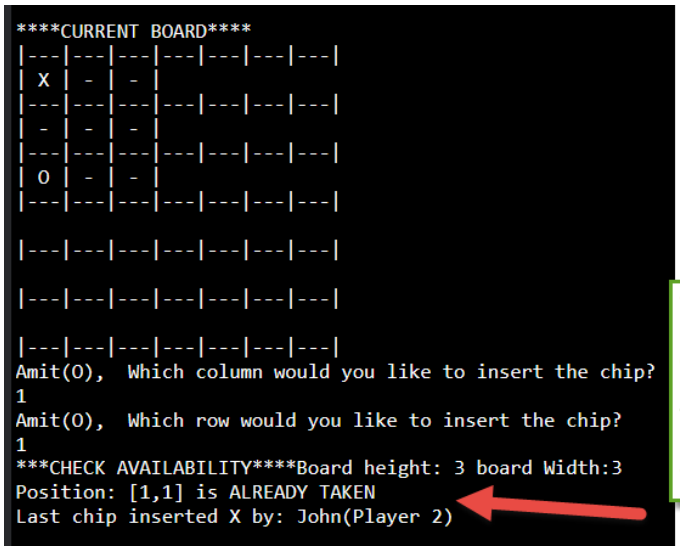


Test case 3d: Apply same logic to all diagonal checks and run code to see if it performs validation: PASS



It performs all diagonal checks as expected

Test 4: Play the actual game and determine winner

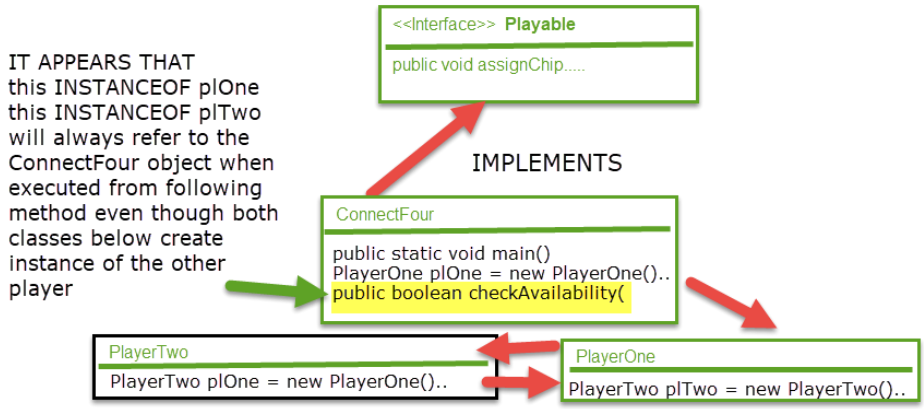


This information is correct, but unfortunately the next turn is passed back to John

This is perfect opportunity to introduce instanceOf

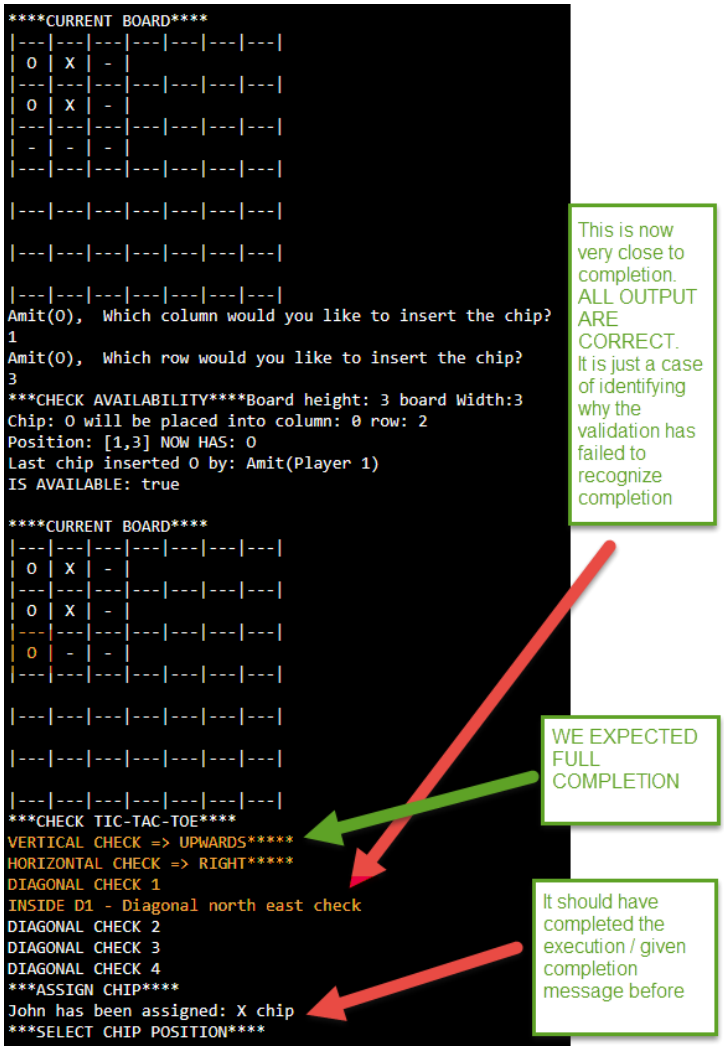
However I have realised, and I did suspect it would challenge my mindset is that instanceOf relates to connectFour

I suspect this is related to the following relationship



So I have performed remediation inline with existing technique of ascertaining last person to place a chip.

Test 5: Play the game to completion



This is now very close to completion. ALL OUTPUT ARE CORRECT. It is just a case of identifying why the validation has failed to recognize completion

WE EXPECTED FULL COMPLETION

It should have completed the execution / given completion message before

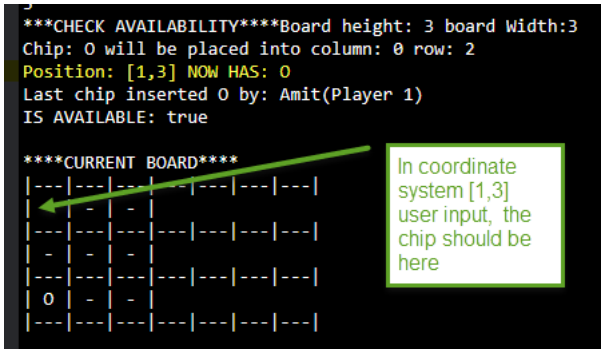
This was relatively straight forward fix.
I had to change all scenarios as such

```
if (sameColourTotal+runningTotalSameColour>=3)
{
```

It now appears the game is complete.

I did however notice one issue as below

Test case 6: Need to adjust logic to reflect coordinate system for the Y axis



This is simply performed by the height of board – user input (row)
End user specifies in non-zero index method.

Once again, this has brought its own mini challenge.. I have narrowed it down as below...

```
//Unlike Connect4, end user would be required to specify two inputs to denote the chip (nought/cross) placement
//We are taking their input to be similar to coordinate system X Y
//hence this is why the indexes below on the board are back to front
//for (int k=(boardHeightZeroIndex); k>=0; k--)

//I have included this logic below since when end user specifies row 3, Java will interpret this as the lowest row on the grid
//If end user specified row 1, Java will interpret at top row.
//The following relationship exists
//User input (row 3) => Java zero index row 2   => It should be zero index row 0
//User input (row 2) => Java zero index row 1   => It should be zero index row 1
//User input (row 1) => Java zero index row 0   => It should be zero index row 2
//The difference absolute value upon subtracting 2 or (boardHeight-1)

System.out.println("board height: " + boardHeight);
System.out.println("row input: " + rowInput);
//{
    if (board[Math.abs(rowInput-(boardHeight-1))][colInput].equals("-"))
    {
        board[Math.abs(rowInput-(boardHeight-1))][colInput]= String.valueOf(chipValue);
```

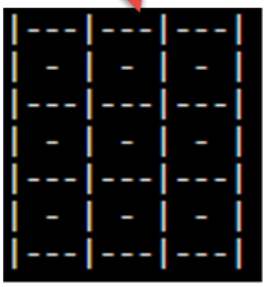
And now I am in a position to insert chip in each position and play the game in a real simulation....
But I need to resolve the board

Test case 7: Resolving the board

```
StringBuilder [] sb = new StringBuilder[8];

for (int i=0; i<sb.length;i++)
{
    if (i%2==0)
    {
        sb[i+j] = new StringBuilder("|---|---|---|");
    }
    else
    {
        sb[i]=new StringBuilder("");

        if (rowOnBoard!=board.length)
        {
            for (int m=0; m<board[0].length;m++)
            {
                if (m==0)
                {
                    sb[i].append("|"+" "+board[rowOnBoard][m]+" |");
                }
                else
                {
                    sb[i].append(""+" "+board[rowOnBoard][m]+" |");
                }
            }
            rowOnBoard++;
        }
    }
}
```

FOR THE MOMENT, I AM STRUGGLING EXTREMELY TO UNDERSTAND THE CODE THAT I HAVE WRITTEN. I WAS PRACTICALLY TWEAKING IT BIT BY BIT TO MEET CONNECT4 BOARD.

THE BEST I CAN DO IS CHANGE THE STRINGBUILDER ARRAY SIZE TO 8.
IT STILL LOOKS PRESENTABLE



Test case 8: Playing the game