

Debugging Complex Methods

by Sophia



WHAT'S COVERED

In this lesson, you will learn about debugging complex methods when using loops. Specifically, this lesson covers:

Table of Contents

- [1. A Method to Detect a Tic-Tac-Toe Win in a Row](#)
- [2. A Method to Detect a Tic-Tac-Toe Win in a Column](#)

1. A Method to Detect a Tic-Tac-Toe Win in a Row

Consider this method intended to determine if there is a win on one of the rows on the Tic-Tac-Toe board.

Here is an attempt at a method called `checkForRowWin()` to detect if a player has marked all three spaces in a given row. To keep things simpler while working through the method, the program relies on a hard-coded array for the board with X's and O's rather than relying on user input:

```
class TicTacToeWin {
    public static void main(String[] args) {
        // Board pattern to test
        // X has a win on the second row (array index 1 is row 2)
        String[][] board = {{" - ", " O ", " X "},
                            {" X ", " X ", " X "},
                            {" O ", " O ", " - "}};
        int winningRow = checkForRowWin(board, 'X');
        if(winningRow > 0) {
            System.out.println("Win on row " + winningRow);
        }
    }

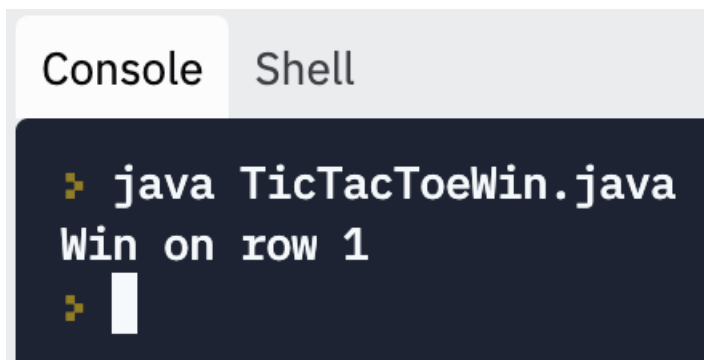
    public static int checkForRowWin(String[][] board, char player) {
        // Variable to track row that holds a win
        int rowNumber = 0;
```

```

for(int row = 0; row <= 3; row++) {
    if(board[row][0].equals(" " + player + " ") ||
       board[row][1].equals(" " + player + " ") ||
       board[row][2].equals(" " + player + " ")) {
        // Add 1 to row number to match human count (starting with 1)
        rowNumber = row + 1;
        break;
    }
}
// If rowNumber > 0, there is a winning row
return rowNumber;
}
}

```

If player X has marked all three spaces in a row, the method returns the number of the row (1 - 3) containing the win; otherwise, it returns 0. Running the code in Replit should show a win on row 2, but this code produces the following result:



```

Console  Shell
➤ java TicTacToeWin.java
Win on row 1
➤

```

To assess the problem, a temporary statement is used. The temporary statement will be removed when the issue has been resolved. When the code detects a winning row, this version of the program will print out the contents of that row.

The lines in bold are the new additions:

➤ EXAMPLE

```

import java.util.Arrays; // Added to print out row

class TicTacToeWin { public static void main(String[] args) {
    // Board pattern to test
    String[][] board = {" - ", " O ", " X "},
                  {" X ", " X ", " X "},
                  {" O ", " O ", " - "};

    int winningRow = checkForRowWin(board, 'X');
    if(winningRow > 0) {
        System.out.println("Win on row " + winningRow);
    }
}
}

```

```

public static int checkForRowWin(String[][] board, char player) {
    // Variable to track row that holds a win
    int rowNumber = 0;
    for(int row = 0; row <= 3; row++) {
        if(board[row][0].equals(" " + player + " ") ||
           board[row][1].equals(" " + player + " ") ||
           board[row][2].equals(" " + player + " ")) {
            rowNumber = row + 1;
            // Temporary statement to print out winning row
            System.out.println(Arrays.toString(board[row]));
            break;
        }
    }
    // If rowNumber > 0, there is a winning row
    return rowNumber;
}
}

```

The added information in the output shows that the code is declaring a win in a row that clearly is not:

ConsoleShell

```

> java TicTacToeWin.java
[ - , 0 , X ]
Win on row 1
>

```



THINK ABOUT IT

There is only one X in the "winning" row. What can be the cause?



TRY IT

Directions: Let's look at the selection statement that is falsely detecting the win:

➞ EXAMPLE

```

if(board[row][0].equals(" " + player + " ") ||
   board[row][1].equals(" " + player + " ") ||
   board[row][2].equals(" " + player + " ")) {

```

The logical or (||) means that a player is declared a winner if the player has at least one space marked in the row. Since a winner has to have all three spaces, the correct choice is a logical and operator (&&).

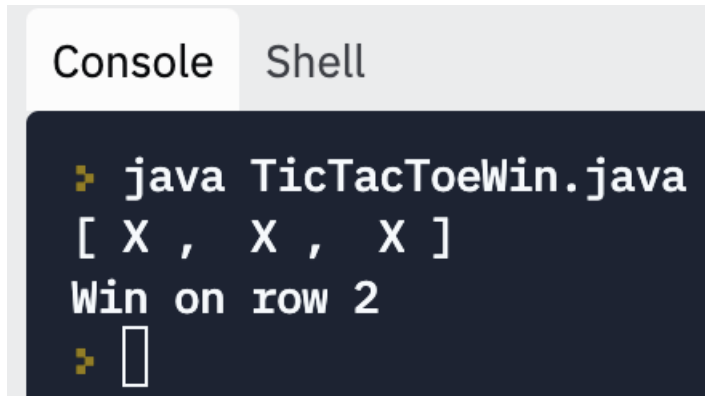
**TRY IT**

Try replacing the code above with the following:

↪ EXAMPLE

```
if(board[row][0].equals(" " + player + " ") &&  
    board[row][1].equals(" " + player + " ") &&  
    board[row][2].equals(" " + player + " ")) {
```

The code now produces the following output:



```
java TicTacToeWin.java  
[ X , X , X ]  
Win on row 2
```

**REFLECT**

When using such a temporary output statement, it is important to display useful content. The code above displays the row when a win has been detected. Would it be useful to have a temporary statement print out the rows where a win is not found?

Now let's try testing with a board that does not contain a winning row.

**TRY IT**

Directions: Change the declaration and initialization of the board array to look like this:

↪ EXAMPLE

```
String[][] board = {{" - ", " O ", " X "},  
                    {" X ", " - ", " X "},  
                    {" O ", " O ", " - "}};
```

Running the code produces this result:

```

> java TicTacToeWin.java
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3
out of bounds for length 3
    at TicTacToeWin.checkForRowWin(TicTacToeWin.java:19)
    at TicTacToeWin.main(TicTacToeWin.java:9)

```



REFLECT

As the error message in the stack trace indicates, the array index 3 is out of range (an array with three rows has indexes 0, 1, and 2). But why is 3 even occurring in the first place?



TRY IT

Directions: Adding another "temporary" output statement in the loop indicating the row being processed (zero-based count) can help us see how the loop is progressing:

➞ EXAMPLE

```

public static int checkForRowWin(String[][] board, char player) {
    // Variable to track row that holds a win
    int rowNumber = 0;
    for(int row = 0; row <= 3; row++) {
        // Temporary statement to display row number as processed
        System.out.println("Checking row " + row + " (0-based count)");
        if(board[row][0].equals(" " + player + " ") ||
           board[row][1].equals(" " + player + " ") ||
           board[row][2].equals(" " + player + " ")) {
            rowNumber = row + 1;
            // Temporary statement to print out winning row
            System.out.println(Arrays.toString(board[row]));
            break;
        }
    }
    // If rowNumber > 0, there is a winning row
    return rowNumber;
}

```

The output now shows how the loop is progressing:

```
➤ java TicTacToeWin.java
Checking row 0 (0-based count)
Checking row 1 (0-based count)
Checking row 2 (0-based count)
Checking row 3 (0-based count)
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out
of bounds for length 3
    at TicTacToeWin.checkForRowWin(TicTacToeWin.java:22)
    at TicTacToeWin.main(TicTacToeWin.java:9)
➤
```



REFLECT

Since the board consists of three rows, the zero-based indexes for the rows should be 0, 1, and 2, but the loop is clearly trying to run four times and access a row with the index 3.



TRY IT

Directions: Look at the line that starts the loop; we can find the problem:

➤ EXAMPLE

```
for(int row = 0; row <= 3; row++) {
```

The value of row should not reach 3, but the incorrect comparison operator `<=` (less than or equal to) causes row to run through the values 0, 1, 2, and 3. Changing the loop's definition to

➤ EXAMPLE

```
for(int row = 0; row < 3; row++) {
```

should fix the problem. If you make the change and run the code, you should get the following results:

```
➤ java TicTacToeWin.java
Checking row 0 (0-based count)
Checking row 1 (0-based count)
Checking row 2 (0-based count)
➤
```



REFLECT

The loop is now running through the correct set of lines.

2. A Method to Detect a Tic-Tac-Toe Win in a Column

It is also important to be able to draft a method that detects a win in one of the columns.



Directions: The code below includes some temporary output to show the contents of the column that is found to contain a win:

```
import java.util.Arrays; // Added to print out row

class TicTacToeWin {
    public static void main(String[] args) {
        // Board pattern to test
        String[][] board = {{" X ", " O ", " X "},
                             {" X ", " O ", " X "},
                             {" O ", " O ", " - "}};
        int winningCol = checkForColumnWin(board, 'O');
        if(winningCol > 0) {
            System.out.println("Win on column " + winningCol);
        }
    }

    public static int checkForColumnWin(String[][] board, char player) {
        // Variable to track row that holds a win
        int colNumber = 0;
        for(int col = 0; col < 3; col++) {
            if(board[col][0].equals(" " + player + " ") &&
               board[col][1].equals(" " + player + " ") &&
               board[col][2].equals(" " + player + " ")) {
                colNumber = col + 1;
                // Temporary statement to print out winning row
                System.out.println("Col:\n"+ board[0][col] + "\n" +
                                   board[1][col] + "\n" + board[2][col]);
                break;
            }
        }
        // If colNumber > 0, there is a winning row
        return colNumber;
    }
}
```

Console Shell

```
➤ java TicTacToeWin.java
➤
```

 REFLECT

Running the code in this form doesn't produce any results. How can we figure out what is going on?

 TRY IT

Directions: Add a temporary output line to show the value of colNumber before the method's return statement. Add the following lines:

➤ EXAMPLE

```
// Temporary output
System.out.println("colNumber: " + colNumber);
return colNumber;
```

Now the result of running the code looks like this:

Console Shell

```
➤ java TicTacToeWin.java
colNumber: 0
➤
```

The added code shows that the value colNumber is 0 when it is returned. Since the code adds 1 to colNumber when a winning column is detected to convert it to a column number as a person would count them, the fact that 0 is returned indicates that no win is found in the columns.

To get a better idea of what is happening as the loop runs through the columns, it is helpful to have a couple added temporary output lines to display the contents of each column as the loop runs.

 TRY IT

Directions: Add these lines before the if() statement in the body of the loop:

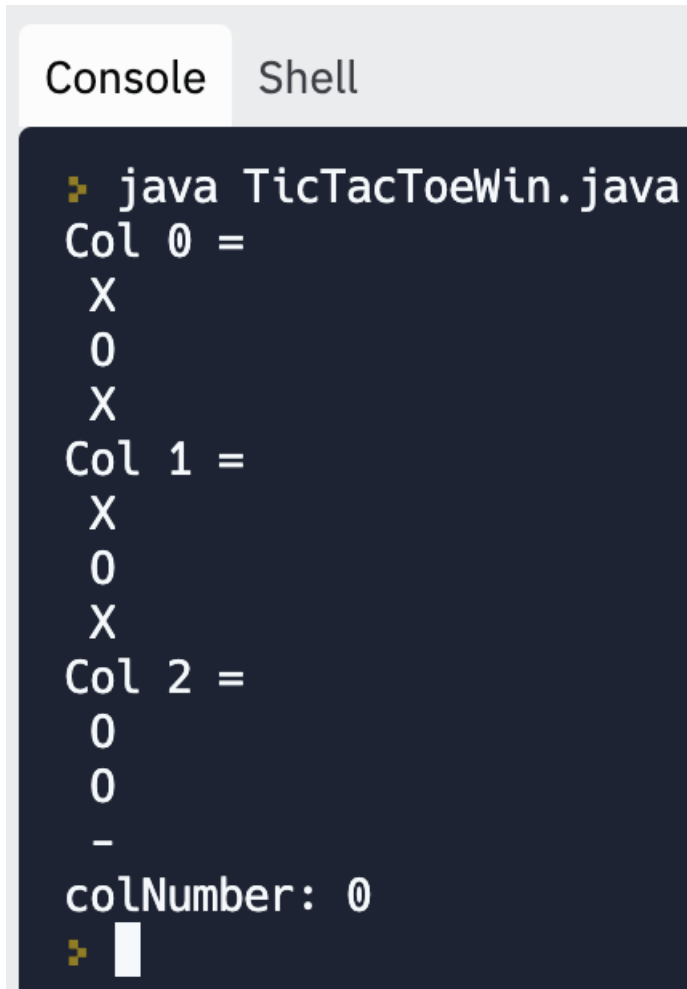
➤ EXAMPLE

```
// 2 lines to display contents of each column
```



```
System.out.println("Col " + col + " = ");
System.out.println(board[col][0] + "\n" + board[col][1] + "\n" + board[col][2]);
```

The added output produces these results:



 REFLECT

When considering this output, it's important to remember that the array representing the board is declared and initialized like this:

➞ EXAMPLE

```
String[][] board = {{ " X ", " O ", " X "},
                    { " X ", " O ", " X "},
                    { " O ", " O ", " - "}};
```

The output lists the contents of the first column (index 0) as X O X, but we can see from the declaration of the board that the first column holds the value X X O. The sequence X O X is found in the first row, not the first column. The output shows the contents of the second column (index 1) as X O X, but the declaration shows that the second column contains O O O. Again, in this case, the X O X is found in a row (the second as well as the first), not a column. The output of the third column (index 2) also shows that the contents in the output don't correspond to the declaration, and the contents displayed for the third column actually correspond to

the third row in the declaration.

Given these facts, it seems clear that the code has somehow confused columns and rows.

Let's look at the selection statement that checks each column for a win:

➞ EXAMPLE

```
if(board[col][0].equals(" " + player + " ") &&
   board[col][1].equals(" " + player + " ") &&
   board[col][2].equals(" " + player + " ")) {
    colNumber = col + 1;
    // Temporary statement to print out winning row
    System.out.println("Col:\n"+ board[0][col] + "\n" +
        board[1][col] + "\n" + board[2][col]);
    break;
}
```



REFLECT

It's important to consider that when working with a two-dimensional array, the first index designates the row, and the second index indicates the column. Clearly, though, this code that checks for a win has mixed up the columns and rows.



TRY IT

Directions: Next, redo this section of the code so that the first index is the row and the second is the column:

➞ EXAMPLE

```
if(board[0][col].equals(" " + player + " ") &&
   board[1][col].equals(" " + player + " ") &&
   board[2][col].equals(" " + player + " ")) {
    colNumber = col + 1;
    // Temporary statement to print out winning row
    System.out.println("Col:\n"+ board[0][col] + "\n" +
        board[1][col] + "\n" + board[2][col]);
    break;
}
```

With the correction to the order of the indices made, the program (which still includes the extra debugging statements) displays the following output when run:

Console

Shell

```
➤ java TicTacToeWin.java
Col 0 =
X
O
X
Col 1 =
X
O
X
Col:
O
O
O
colNumber: 2
Win on column 2
➤ □
```



REFLECT

The third column (index 2) is indeed the winning column with three O's.



TRY IT

Directions: Remove the extra temporary output statements; the correct version of the draft method looks like this:

```
public static int checkForColumnWin(String[][] board, char player) {
    // Variable to track row that holds a win
    int colNumber = 0;
    for(int col = 0; col < 3; col++) {
        if(board[0][col].equals(" " + player + " ") &&
           board[1][col].equals(" " + player + " ") &&
           board[2][col].equals(" " + player + " ")) {
            // Correct for human count (starting with 1, not 0)
            colNumber = col + 1;
            break;
        }
    }
    // If colNumber > 0, there is a winning row
```

```
return colNumber;  
}
```



REFLECT

Removing the temporary output statements gets rid of the extra output that was for the developer but not the end user. After removing the extra output, it is important to make sure that the program compiles and runs correctly.



SUMMARY

In this lesson, we have looked at a couple drafts of **methods to detect a win in a row** or **methods to detect a win in a column** in the Tic-Tac-Toe game. You learned that the initial draft versions contained errors, and we have seen how temporary output statements can be used to show how the process of checking the rows or columns is progressing. You also learned that while working on these methods in isolation forms, the rest of the code has made it possible to work out how to handle win detection. In the final lesson for this challenge, we will get these methods worked into the complete game and add the other necessary methods to get the game working correctly.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/