Revisiting the Tic-Tac-Toe Program

by Sophia

WHAT'S COVERED

In this lesson, you will expand on the Tic-Tac-Toe program using loops. Specifically, this lesson covers:

Table of Contents

1. Tic-Tac-Toe With Loops

This is a great time to revisit the Tic-Tac-Toe game that you worked on at the end of the last challenge. If you recall, there were many different instances where loops could have been used. Let's first look at the code that we had previously ended with.

TRY IT

Directions: If you don't have an existing Repl (file) in Replit (TicTacToe.java), enter in the following code to reflect where the Tic-Tac-Toe program ended in tutorial 2.1.8:

```
import java.util.Arrays;
import java.util.Scanner;

public class TicTacToe {

  public static void main(String[] args) {
    String[][] board = {{" - ", " - ", " - "},
                    {" - ", " - ", " - "},
                    {" - ", " - ", " - "}};

    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    Scanner input = new Scanner(System.in);
    int col;
    int row;

    // X's 1st turn
    System.out.print("X - Select column (1 - 3) & select row (1 - 3) ");
    System.out.print("separated by a space: ");
    col = input.nextInt();
    row = input.nextInt();
    if(board[row - 1][col - 1].equals(" - ")) {
      board[row - 1][col - 1] = " X ";
    }
    else {
      System.out.println("Sorry, that spot is taken.");
    }
    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    // O's 1st turn
    System.out.print("O - Select column (1-3) & select row (1-3) ");
    System.out.print("separated by a space: ");
    col = input.nextInt();
    row = input.nextInt();
    if(board[row - 1][col - 1].equals(" - ")) {
      board[row - 1][col - 1] = " O ";
    }
    else {
      System.out.println("Sorry, that spot is taken.");
    }
    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    // X's 2nd turn
    System.out.print("X - Select column (1 - 3) & select row (1 - 3) ");
    System.out.print("separated by a space: ");
    col = input.nextInt();
    row = input.nextInt();
    if(board[row - 1][col - 1].equals(" - ")) {
      board[row - 1][col - 1] = " X ";
    }
    else {
      System.out.println("Sorry, that spot is taken.");
    }
    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");


    // O's 2nd turn
    System.out.print("O - Select column (1-3) & select row (1-3) ");
    System.out.print("separated by a space: ");
    col = input.nextInt();
    row = input.nextInt();
    if(board[row - 1][col - 1].equals(" - ")) {
      board[row - 1][col - 1] = " O ";
    }
    else {
      System.out.println("Sorry, that spot is taken.");
    }
    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    // X's 3rd turn
    System.out.print("X - Select column (1 - 3) & select row (1 - 3) ");
    System.out.print("separated by a space: ");
    col = input.nextInt();
    row = input.nextInt();
    if(board[row - 1][col - 1].equals(" - ")) {
      board[row - 1][col - 1] = " X ";
    }
    else {
      System.out.println("Sorry, that spot is taken.");
    }
    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    // O's 3rd turn
    System.out.print("O - Select column (1-3) & select row (1-3) ");
    System.out.print("separated by a space: ");
    col = input.nextInt();
    row = input.nextInt();
    if(board[row - 1][col - 1].equals(" - ")) {
      board[row - 1][col - 1] = " O ";
    }
    else {
      System.out.println("Sorry, that spot is taken.");
    }
    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    // X's 4th turn
    System.out.print("X - Select column (1 - 3) & select row (1 - 3) ");
    System.out.print("separated by a space: ");
    col = input.nextInt();
    row = input.nextInt();
    if(board[row - 1][col - 1].equals(" - ")) {
      board[row - 1][col - 1] = " X ";
    }
    else {
      System.out.println("Sorry, that spot is taken.");
    }
    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    // O's 4th turn
    System.out.print("O - Select column (1-3) & select row (1-3) ");
    System.out.print("separated by a space: ");
    col = input.nextInt();
    row = input.nextInt();
    if(board[row - 1][col - 1].equals(" - ")) {
      board[row - 1][col - 1] = " O ";
    }
    else {
      System.out.println("Sorry, that spot is taken.");
    }
    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    // X's 5th turn
    System.out.print("X - Select column (1 - 3) & select row (1 - 3) ");
    System.out.print("separated by a space: ");
    col = input.nextInt();
    row = input.nextInt();
    if(board[row - 1][col - 1].equals(" - ")) {
      board[row - 1][col - 1] = " X ";
    }
    else {
      System.out.println("Sorry, that spot is taken.");
```

```
    }
    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");
  }
}
```

REFLECT

When you last saw this code, you were asked to reflect on common steps in the turns. What common steps did you come up with? Since we will next turn to adding loops to handle repetitive steps in the code, it's good to have specific ideas about what those steps are.

TRY IT

Directions: Now that you have the Tic-Tac-Toe program where you left off in the previous challenge, go ahead and run it a few times to get reacquainted with its process.

REFLECT

Now that you've had the chance to rerun the program, do you observe anything that would enhance its functionality?

In the current state of coding, you can quickly determine that in some situations, we have some repetitiveness. Notice that you are constantly prompting each user for their selected position (O's third move, X's fifth move, etc.). You could loop those inputs for both the X and O players as part of the body of the loop. You will keep X player's input at the start where we don't have to determine if the spot has been taken. That essentially is letting the X user go first before the looping begins.

TRY IT

Directions: Adding a new element (loop) to the program code, comments have been used to call it out. Add the looping for X and O player's section for repeat player position input below. Then run the program to see the functionality:

```
import java.util.Arrays;
import java.util.Scanner;

public class TicTacToeLoop {

  public static void main(String[] args) {
    String[][] board = {{" - ", " - ", " - "},
                        {" - ", " - ", " - "},
                        {" - ", " - ", " - "}};

    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    Scanner input = new Scanner(System.in);
    int col;
    int row;

    // for loop to provide 5 rounds of turns
    for(int turn = 0; turn < 5; turn++) {
      // X's turn
      System.out.print("X - Select column (1 - 3) & select row (1 - 3) ");
      System.out.print("separated by a space: ");
      col = input.nextInt();
      row = input.nextInt();
      if(board[row - 1][col - 1].equals(" - ")) {
        board[row - 1][col - 1] = " X ";
      }
      else {
        System.out.println("Sorry, that spot is taken.");
      }
      System.out.println("\t" + Arrays.toString(board[0]));
      System.out.println("\t" + Arrays.toString(board[1]));
      System.out.println("\t" + Arrays.toString(board[2]) + "\n");

      // If 5th turn, end loop after X's turn - only 9 spaces to fill
      if(turn == 4) {
        break;
      }

      // O's turn
      System.out.print("O - Select column (1-3) & select row (1-3) ");
      System.out.print("separated by a space: ");
      col = input.nextInt();
      row = input.nextInt();
      if(board[row - 1][col - 1].equals(" - ")) {
        board[row - 1][col - 1] = " O ";
      }
      else {
        System.out.println("Sorry, that spot is taken.");
      }
      System.out.println("\t" + Arrays.toString(board[0]));
      System.out.println("\t" + Arrays.toString(board[1]));
      System.out.println("\t" + Arrays.toString(board[2]) + "\n");
    }
  }
}
```

REFLECT

Did you notice any change to the functionality of the program? Probably not, but using a for loop has brought the code down from 150+ lines to 57 lines of code and has made it much clearer.

TRY IT

Directions: Change the loop initializing row to include nine iterations instead of five. Change the following:

EXAMPLE

From:
```
for(int turn = 0; turn < 5; turn++) {
```

To:
```
for(int turn = 0; turn < 9; turn++) {
```

REFLECT

While this change increases the number of times the loop will run, how could this change help simplify the code?

TRY IT

Directions: Using a char variable (player), we can use the rest of the original body of the loop within the original “O move” section for both players X and O now. Before the start of the loop, declare the player variable like this:

```
char player;
```

For iteration 0 and the following even iterations, it is X's turn. The odd-numbered iterations are O's turn. We can use this code in the for loop (at the start or top of the loop) to set the player variable to X or O, as appropriate. This code should be the first statements in the body of the for loop.

HINT

There is a full listing of the code after the discussion of the sections to be added or changed, if you have questions about where the changes should be made.

EXAMPLE

```
if(turn % 2 == 0) {
  player = 'X';
}
else {
  player = 'O';
}
```

We will now include the player variable in the body of the loop to indicate whether it's X's or O's turn.

TRY IT

Directions: Change these following rows:

EXAMPLE

```
From:
System.out.print("X - Select row (1 - 3) & select column (1 - 3) ");
System.out.print("separated by a space: ");
row = input.nextInt();
col = input.nextInt();
if(board[row - 1][col - 1].equals(" - ")) {
  board[row - 1][col - 1] = " X ";
}

To:
System.out.print(player + " - Select row (1 - 3) & select column (1 - 3) ");
System.out.print("separated by a space: ");
row = input.nextInt();
col = input.nextInt();
if(board[row - 1][col - 1].equals(" - ")) {
  board[row - 1][col - 1] = " " + player + " ";
}
```

Since this code handles the prompt and input for both X's and O's turns, remove the previous code that handled O's turn.

REFLECT

Using the same lines of code to handle both players' turns has increased the level of abstraction in the code by using a single variable to prompt the correct player. How does such abstraction help programmers write code more concisely and efficiently?

TRY IT

Directions: Make sure you have made the changes discussed above so your program will look like the code below. We have added some comments to the lines of code that were added or changed. Once you have verified the changes, go ahead and run the program. From the user's perspective, the program should function the same as the previous versions:

```
import java.util.Arrays;
import java.util.Scanner;

public class TicTacToeLoop {

  public static void main(String[] args) {
    String[][] board = {{" - ", " - ", " - "},
                        {" - ", " - ", " - "},
                        {" - ", " - ", " - "}};

    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");
    Scanner input = new Scanner(System.in);
    int col;
    int row;
    // Variable to keep track of current player
    char player;

    // for loop to provide total of 9 turns
    for(int turn = 0; turn < 9; turn++) {
      // If an even turn number, player is X
      if(turn % 2 == 0) {
        player = 'X';
      }
      // Otherwise, odd turns are O's turns
      else {
        player = 'O';
      }

      System.out.print(player + " - Select row (1 - 3) & select column (1 - 3) ");
      System.out.print("separated by a space: ");
      row = input.nextInt();
      col = input.nextInt();
      if(board[row - 1][col - 1].equals(" - ")) {
        // Use player to provide character (X or O)
        board[row - 1][col - 1] = " " + player + " ";
      }
      else {
        System.out.println("Sorry, that spot is taken.");
      }
      System.out.println("\t" + Arrays.toString(board[0]));
      System.out.println("\t" + Arrays.toString(board[1]));
      System.out.println("\t" + Arrays.toString(board[2]) + "\n");
    }
  }
}
```

REFLECT

When changes are made in code to improve the structure or performance of the program that are not visible to the end user, the process of making such changes is called "refactoring." How does such refactoring fit into the process of creating an application?

2. Adding Validation

These changes in the code have really helped simplify the code. However, there is more that can be done to make the game even better. The next thing that you can do is do some error checking or **validation** based on the row and column that was entered by the player.

You will want to check two separate things:

1. Ensure that the value being entered by the player is between 1 and 3. Remember, our two-dimensional list (2D) is three columns by three rows.
2. Ensure that if the position in the 2D list has an entry already there, we want to inform the player to choose again. Right now if a player selects a position that is already taken, they lose their turn.

2a. Solution for Validating the User Input Value

Item #1 can be solved above using a while loop to check for positions that are not valid. For the loop to work correctly, be sure to initialize the row and col variables when they are declared:

↪ EXAMPLE

```
int col = 0;
int row = 0;
```

TRY IT

Directions: Set up a while loop like this:

↪ EXAMPLE

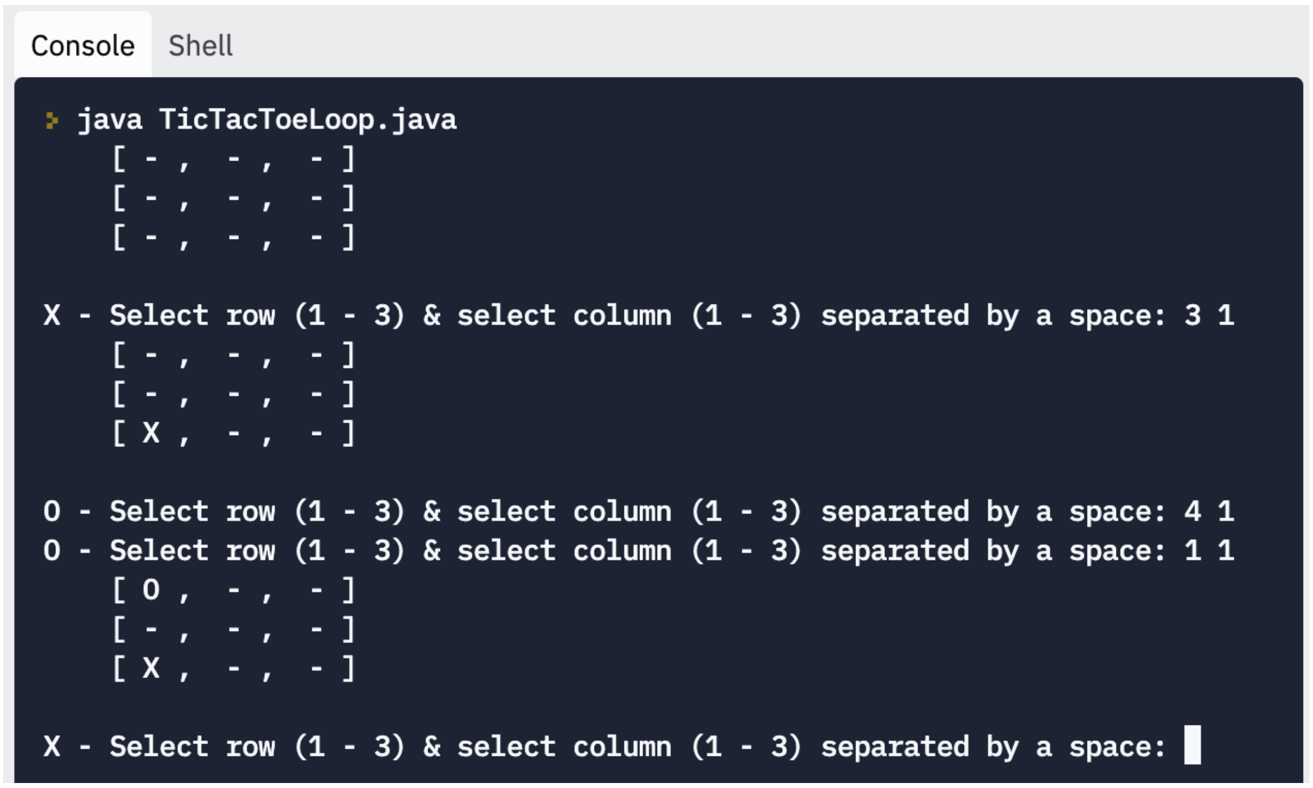
```
  while(col < 1 || col > 3 || row < 1 || row > 3){
      System.out.print(player + " - Select row (1 - 3) & select column (1 - 3) ");
      System.out.print("separated by a space: ");
      row = input.nextInt();
      col = input.nextInt();
    }
```

Directions: At the bottom of the loop, set the values of col and row back to 0 so that the while loop will run and prompt the user for input (and validate the new input):

↪ EXAMPLE

```
col = 0;
row = 0;
```

In the code above, the while loop's condition states that as long as the variable col's value is less than 1 or greater than 3, the statements in the body of the loop would be repeated. If the entry from the user for col was less than 1 or larger than 3, you will output to the user that the column has to be between 1 and 3.



```
> java TicTacToeLoop.java
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]

X - Select row (1 - 3) & select column (1 - 3) separated by a space: 3 1
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]
    [ X ,  - ,  - ]

O - Select row (1 - 3) & select column (1 - 3) separated by a space: 4 1
O - Select row (1 - 3) & select column (1 - 3) separated by a space: 1 1
    [ O ,  - ,  - ]
    [ - ,  - ,  - ]
    [ X ,  - ,  - ]

X - Select row (1 - 3) & select column (1 - 3) separated by a space: 
```

REFLECT

The loop that handles the turns is a for loop because the maximum number of turns is a known value and does not change. The validation loop needs to be a while, though. Why is this the case? (Why wouldn't a for loop be the right choice for the validation loop?)

Here is the full code at this point; the solutions for item 1 (while loops from column and row input) are commented below.

Notice that we set the col and row to 0 outside of the while loop so with each iteration, the col and row will be reset to ensure that the user enters in an updated value each time:

```
import java.util.Arrays;
import java.util.Scanner;

public class TicTacToeLoop {

  public static void main(String[] args) {
    String[][] board = {{" - ", " - ", " - "},
                    {" - ", " - ", " - "},
                    {" - ", " - ", " - "}};

    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    Scanner input = new Scanner(System.in);
    // Initialize col & row so that the input validation works
    int col = 0;
    int row = 0;
    // Variable to keep track of current player
    char player;

    // for loop to provide total of 9 turns
    for(int turn = 0; turn < 9; turn++) {
      // If an even turn number, player is X
      if(turn % 2 == 0) {
        player = 'X';
      }
      // Otherwise, odd turns are O's turns
      else {
        player = 'O';
      }
      // Check if col or row is less than 1 or greater than 3
      while(col < 1 || col > 3 || row < 1 || row > 3){
        System.out.print(player + " - Select row (1 - 3) & select column (1 - 3) ");
        System.out.print("separated by a space: ");
        row = input.nextInt();
        col = input.nextInt();
      }
      if(board[row - 1][col - 1].equals(" - ")) {
        // Use player to provide character (X or O)
        board[row - 1][col - 1] = " " + player + " ";
      }
      else {
        System.out.println("Sorry, that spot is taken.");
      }
      // Reset col and row to 0 so the validation loop runs for next turn
      col = 0;
      row = 0;
      System.out.println("\t" + Arrays.toString(board[0]));
      System.out.println("\t" + Arrays.toString(board[1]));
      System.out.println("\t" + Arrays.toString(board[2]) + "\n");
    }
  }
}
```

TRY IT

Directions: Try running this program with the changes above and test for outside positions (1< or >3). Did the program produce the intended results?

2b. Solution for Validating the Position in the 2D List Has an Entry

There is already a selection statement that checks if the position on the board is open or already taken. If it is taken, the code prints the appropriate message:

```
if(board[row - 1][col - 1].equals(" - ")) {
  // Use player to provide character (X or O)
  board[row - 1][col - 1] = " " + player + " ";
}
else {
  System.out.println("Sorry, that spot is taken.");
}
```

To keep the player from losing a turn if the position is already taken, we'll add a statement to the else block to roll the for loop's variable back by one so that the player's turn repeats:

↪ EXAMPLE

```
turn--;
```

The board should not be printed out again unless the selection was valid and the position was open. Input the code for printing out the board in the if() block that checks if the value at the position is equal to " - ".

This section of the code will end up like this:

↪ EXAMPLE

```
if(board[row][col].equals(" - ")) {
  // Use player to provide character (X or O)
  board[row][col] = " " + player + " ";
  System.out.println("\t" + Arrays.toString(board[0]));
  System.out.println("\t" + Arrays.toString(board[1]));
  System.out.println("\t" + Arrays.toString(board[2]) + "\n");
}
else {
  // If position already taken, print message
  System.out.println("Sorry, that spot is taken.");
  // Roll loop var back by 1 so that the turn repeats
  turn--;
}
```

TRY IT

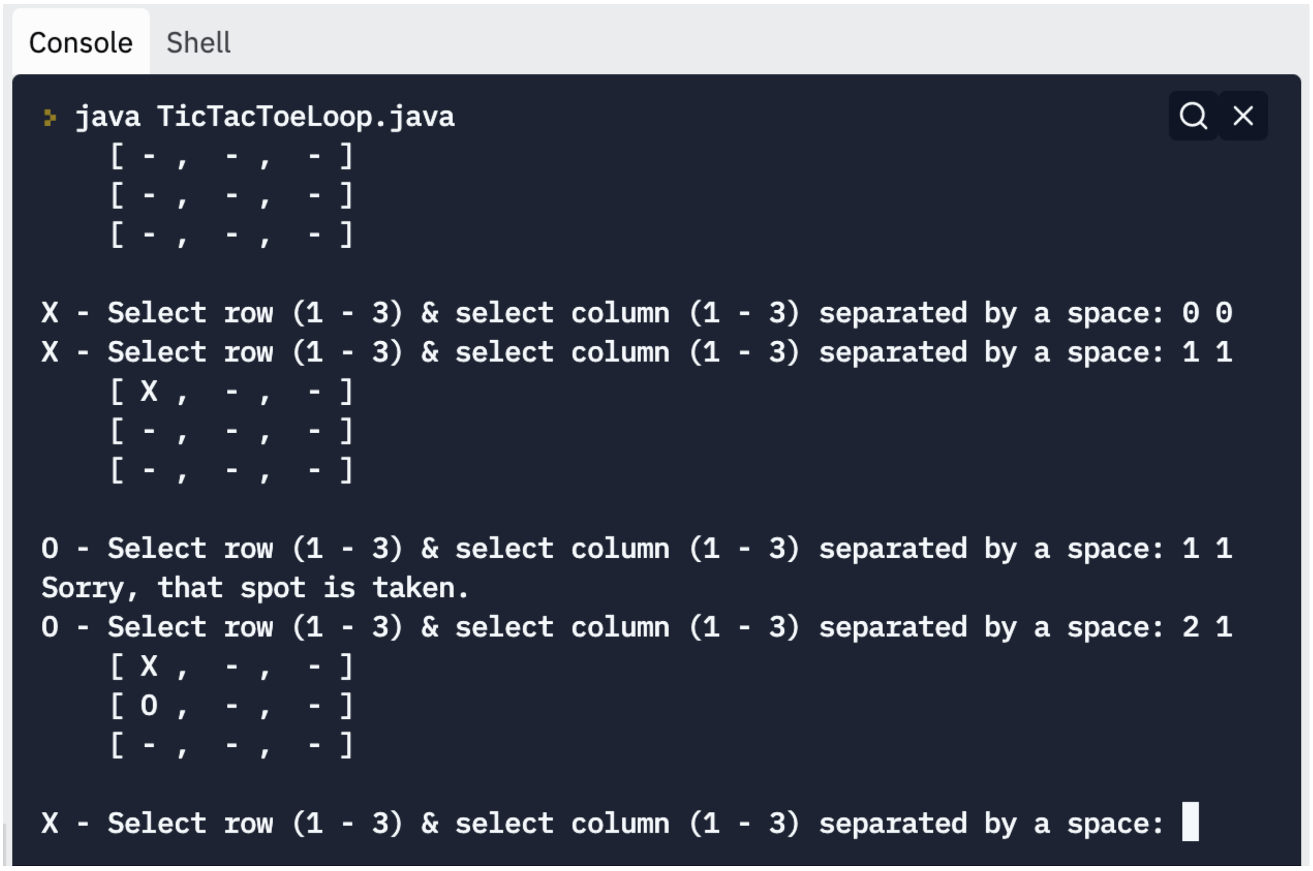
Directions: The complete program should now look like the following. Go ahead and make sure that your code matches this (or type the code into a file named TicTacToeLoop.java now):

```
import java.util.Arrays;
import java.util.Scanner;
```

```
public class TicTacToeLoop {

  public static void main(String[] args) {
    String[][] board = {{" - ", " - ", " - "},
                        {" - ", " - ", " - "},
                        {" - ", " - ", " - "}};

    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    Scanner input = new Scanner(System.in);
    // Initialize col & row so that the input validation works
    int col = 0;
    int row = 0;
    // Variable to keep track of current player
    char player;

    // for loop to provide total of 9 turns
    for(int turn = 0; turn < 9; turn++) {
      // If an even turn number, player is X
      if(turn % 2 == 0) {
        player = 'X';
      }
      // Otherwise, odd turns are O's turns
      else {
        player = 'O';
      }
      // Check if col or row is less than 1 or greater than 3
      while(col < 1 || col > 3 || row < 1 || row > 3){
        System.out.print(player + " - Select row (1 - 3) & select column (1 - 3) ");
        System.out.print("separated by a space: ");
        row = input.nextInt();
        col = input.nextInt();
      }
      if(board[row - 1][col - 1].equals(" - ")) {
        // Use player to provide character (X or O)
        board[row - 1][col - 1] = " " + player + " ";
        System.out.println("\t" + Arrays.toString(board[0]));
        System.out.println("\t" + Arrays.toString(board[1]));
        System.out.println("\t" + Arrays.toString(board[2]) + "\n");
      }
      else {
        // If position already taken, print message
        System.out.println("Sorry, that spot is taken.");
        // Roll loop var back by 1 so that the turn repeats
```

```
          turn--;
        }
        // Reset col and row so that validation loop runs correctly
        col = 0;
        row = 0;
      }
    }
  }
```

Running this code should produce results like this:


Console Shell

```
> java TicTacToeLoop.java
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]

X - Select row (1 - 3) & select column (1 - 3) separated by a space: 0 0
X - Select row (1 - 3) & select column (1 - 3) separated by a space: 1 1
    [ X ,  - ,  - ]
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]

O - Select row (1 - 3) & select column (1 - 3) separated by a space: 1 1
Sorry, that spot is taken.
O - Select row (1 - 3) & select column (1 - 3) separated by a space: 2 1
    [ X ,  - ,  - ]
    [ O ,  - ,  - ]
    [ - ,  - ,  - ]

X - Select row (1 - 3) & select column (1 - 3) separated by a space:
```

REFLECT

When running the program, think about the relationship between the "outer" for loop that controls the turns and the while loop that handles the validation. Which parts of the output are controlled by which loop?

BRAINSTORM

Go ahead and try running this program again with the additional logic to check the selected spot and test some cases of positions that are already taken. There is still room for improvement, such as checking if the player has won or not. We'll get further into that in the next challenge with functions and methods.

TERM TO KNOW

Validation
The process in a program of checking that input is in a valid range and appropriate in the current state of the program.

SUMMARY

In this lesson, you improved the **Tic-Tac-Toe program by using loops**for player input. Using loops, you greatly reduced the large blocks of repetitive code from our first version of the game. You then created nested loops to check for input **position validation** within the game board and whether or not the chosen position was already taken. If either validation was invalid, an output message was sent to the player, and they were asked to try again.

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/

TERMS TO KNOW

Validation

The process in a program of checking that input is in a valid range and appropriate in the current state of the program.