Tic-Tac-Toe Program

by Sophia

WHAT'S COVERED

In this lesson, you will learn about and how to create a program that uses arrays and applies earlier concepts. Specifically, this lesson covers:

1. Review of Arrays

As you learned in tutorial 2.1.1, an array is a special type of variable that represents multiple values all of the same data type. You learned about working with single-dimensional arrays and two-dimensional arrays. This includes acknowledging the existence of three-dimensional arrays, though we are not working with them. Remember that arrays, unlike collections, have fixed sizes. Arrays are often useful for modeling simple structures in the real world that are characterized by sets of values, as long as the values are the same type of data.

1a. One-Dimensional Arrays

A single-dimensional array is like a row of numbered boxes or slots, where each space allows the storing one value of the data type declared for the array. These spaces (or boxes or slots) in the array are called elements. Each element has an index, an integer value, that identifies it in the array and indicates its position in the array. The first element in an array has the index 0, and the last element in the array has an index equal to the declared size of the array minus 1.

HINT

Remember that unlike a collection, the size of an array is fixed and cannot easily change. The values in the elements in the array can, however, be changed.

1b. Two-Dimensional Arrays

While a single-dimensional array can be thought of as a single row of elements, a two-dimensional array can be pictured as a grid or checkerboard with rows and columns. A two-dimensional array is specified using the number of rows and the number of columns. As with a single-dimensional array, the size of the two-dimensional array (the number of rows and the number of columns) is fixed when the array is declared. Elements in a two-dimensional array are specified using two indices: the index for the row and the index for the column. The value of a single-dimensional array can be changed, and the value of any element can be changed as well.

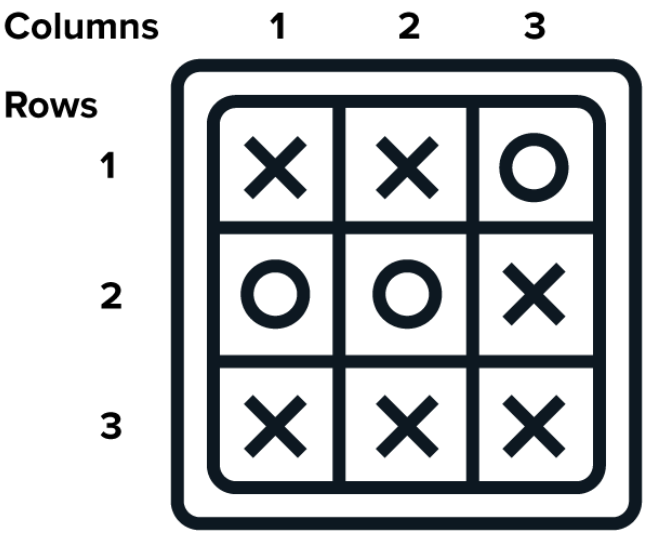
HINT

When programming in Java, it is helpful to think of a two-dimensional array as being an array of arrays. Unlike a single-dimensional array which consists of an integer or string values, think of a two-dimensional array as a series of elements in an array. Each array contains a one-dimensional array. Each nested array is a row in the two-dimensional array.

2. The Start of the Tic-Tac-Toe Game

Picture a Tic-Tac-Toe board. As you look at it, you will see that it has three rows, and each row has three squares (columns).

The size of the board is fixed and does not vary:



Each of the squares can be empty, or contain an X or an O. The position of each square never changes, so the order does matter. The contents of each of the squares can change from empty to either an X or an O, but then change no further. Given the data structures you have worked with, consider which is the best type to represent a row in a Tic-Tac-Toe board. Keep in mind that a row has exactly three squares. Each square contains information that can change and the order matters.

Since each row consists of three squares, it can be viewed as a single-dimensional array. There are three rows, so the board can be seen as three single-dimensional arrays joined into a two-dimensional array.

TRY IT

Directions: Model a row like this:

EXAMPLE

```
String[] row0 = {" - ", " - ", " - "};
```

The - indicates that the cell is empty (has been filled with an X or an O).

Directions: Given that a Tic-Tac-Toe board has three rows, we can now define the rows like this:

EXAMPLE

```
String[] row0 = {" - ", " - ", " - "};
String[] row1 = {" - ", " - ", " - "};
String[] row2 = {" - ", " - ", " - "};
```

Directions: Model our board like this using the previous rows and join them together into a two-dimensional array (an array of single-dimensional arrays):

EXAMPLE

```
String[][] board = new String[3][3];
board[0] = row0;
board[1] = row1;
board[2] = row2;
```

REFLECT

This code is not a complete program, but it is a first attempt at showing how a Tic-Tac-Toe board can be represented as a two-dimensional array. Each row in the game is a one-dimensional array.

STEP BY STEP

1. The first step is Board Creation. Instead of this structure, we can also establish a two-dimensional array to define it. This approach would have the same result, and it could be declared like the following:

Enter the following code into the IDE to start the build process for the Tic-Tac-Toe board:

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

Use this nested approach, since it makes the data easier to access and reflects the real-world behavior of the Tic-Tac-Toe board. There are three columns with three elements in each row in this list, as it resembles the board.

For now, make this game a bit easier by allowing players to alternate and choose the position. Once they have selected the position, they will be able to place the X or O in that position as long as the position still has a "-" (hyphen). Use the hyphen as an empty position.

At this point, do not introduce a checking function to determine if anyone has won. For now, this board is just meant to be a replacement for drawing out the board on paper. As you work through this, think about ways to improve this process going forward.

Directions: Start by setting up the board and printing it out. Use the println() and Arrays.deepToString() methods to see what this board looks like on screen:

```
import java.util.Arrays;

public class TicTacToe {

  public static void main(String[] args) {
    String[][] board = {{" - ", " - ", " - "},
                        {" - ", " - ", " - "},
                        {" - ", " - ", " - "}};
    System.out.println(Arrays.deepToString(board));
  }
}
```

The results show a 2D list as the output:

```
[[ - ,  - ,  - ], [ - ,  - ,  - ], [ - ,  - ,  - ]]
```

Does that look like a Tic-Tac-Toe board? Not a traditional board, though. The deepToString() method doesn't keep the formatting from the definition. Although this is our board, it would be a bit difficult to tell who won visually. To help with that, we want to output each row separately.

Directions: Now try the code below with each row being separately outputted to the screen:

```
import java.util.Arrays;

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]));
  }
}
```

The results should show the initial Tic-Tac-Toe board as the output:

```
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]
```

REFLECT

That's a bit better. However, what other ways could this be improved as you move into the next step? This small but complete program shows an evolution in design over the previous version by declaring the game board as a single two-dimensional array. Look at both versions of the code again to appreciate how the code has been refined.

2. The second step is to prompt the user to enter in a value between 1-3 for the column and then again (1-3) for the row. Although the array starts with an index of 0 and goes to 2 (three elements), that would be confusing to the end user, which is why we will let them select between 1 and 3.

TRY IT

Directions: Enter the code to output directions to the X user for their selection:

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

public class TicTacToe {

  public static void main(String[] args) {
    int col;
```

```
    int row;
    Scanner input = new Scanner(System.in);
    String[][] board = {{" - ", " - ", " - "},
                        {" - ", " - ", " - "},
                        {" - ", " - ", " - "}};
    // The leading tab chars (\t) indent the board
    System.out.println("\t" + Arrays.toString(board[0]));
    System.out.println("\t" + Arrays.toString(board[1]));
    // The \n adds a blank line below the board
    System.out.println("\t" + Arrays.toString(board[2]) + "\n");

    // Display prompt text
    System.out.print("X - Select row (1 - 3) & select column (1 - 3) ");
    // Space separates row & column
    System.out.print("separated by a space: ");
    col = input.nextInt();
    row = input.nextInt();
  }
}
```

Running the code above should produce output screen that shows the selection prompt.

```
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]

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

REFLECT

An X can now be placed in that position; output the board again. Remember that the user has entered values in the range from 1 to 3, but the indices of the array run from 0 to 2. The code will need to subtract 1 from the row and column to update the correct square on the board.

TRY IT

Directions: Add the following code to place the X in the row and column that has been identified:

```
board[row - 1][col - 1] = " X ";
System.out.println("\nTic-Tac-Toe Board:\n");
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");
```

Now test what you have so far. Make sure your code looks like the code snippet below, then run the program:

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

public class TicTacToe {

  public static void main(String[] args) {
    int col;
    int row;
    Scanner input = new Scanner(System.in);

    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");

          // Display prompt text
    System.out.print("X - Select row (1 - 3) & select column (1 - 3) ");
          // Space separates row & column
    System.out.print("separated by a space: ");
    row = input.nextInt();
    col = input.nextInt();
    // Mark the requested square. Subtract 1 to get correct square
    board[row - 1][col - 1] = " X ";
    // Print updated 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");
  }
}
```

The results should show the first move by player X as the output.:

```
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]

X - Select row (1 - 3) & select column (1 - 3) separated by a space: 2 1
    [ - ,  - ,  - ]
    [ X ,  - ,  - ]
    [ - ,  - ,  - ]
```

REFLECT

Did you (as player X) get your X in the column/row you selected?

Next, prompt the player playing O to go next. Use the same approach with a couple substitutions:

TRY IT

Directions: Try adding the code for the second player to enter their position:

```
System.out.print("O - Select row (1-3) & select column (1-3) ");
System.out.print("separated by a space: ");
row = input.nextInt();
col = input.nextInt();
// Mark the requested square. Subtract 1 to get correct square
board[row - 1][col - 1] = " O ";
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

While entering this code, you may have noticed a potential problem. What if player O enters a position that player X has already chosen?

You do not want to overwrite that position. Rather, you will just say that the user ends up forfeiting their turn. The game then moves onto the next player. This is certainly not ideal. It is important to think about what you ideally want as you move forward regarding if you want the user to keep selecting a spot until they choose one that is available. Look for that type of correction in future tutorials. For now, check if the spot is empty by looking for the "–" (hyphen) character. If the selection is the "-" character, we'll replace it with an O, and if it wasn't, we'll let the user know that spot was already taken, and the other player selects.

TRY IT

Directions: Enter the code to check if the space has been filled or not:

```
if(board[row - 1][col - 1].equals(" - ")) {
  board[row - 1][col - 1] = " O ";
}
else {
  System.out.println("Sorry, that spot is taken.");
}
```

3. The last step is to repeat this structure seven more times since you have two entries complete and have nine total boxes to fill. This can get a bit busy with the number of times you repeat it. It would be beneficial to add comments to indicate each move.

TRY IT

Directions: Enter the next seven attempts to complete the Tic-Tac-Toe game. See the comments to separate each turn:

```
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 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 ";
    }
    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 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] = " 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 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 ";
}
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 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] = " 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 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 ";
}
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 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] = " 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 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 ";
}
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 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] = " 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 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 ";
    }
    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

This version of the program includes code to handle all of the turns in the game. To get ready for the next version of the Tic-Tac-Toe game, think about the common steps involved in each turn.

We're now ready to test it to see how we did!

TRY IT

Directions: Run the Tic-Tac-Toe Program and enter choices for players X and O:

The result should look like this with a complete Tic-Tac-Toe game:

```
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]
    [ - ,  - ,  - ]

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: 2 1
    [ X ,  - ,  - ]
    [ O ,  - ,  - ]
    [ - ,  - ,  - ]

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

O - Select row (1-3) & select column (1-3) separated by a space: 1 2
    [ X ,  O ,  - ]
    [ O ,  - ,  - ]
    [ X ,  - ,  - ]

X - Select row (1 - 3) & select column (1 - 3) separated by a space: 2 2
    [ X ,  O ,  - ]
    [ O ,  X ,  - ]
    [ X ,  - ,  - ]

O - Select row (1-3) & select column (1-3) separated by a space: 3 2
    [ X ,  O ,  - ]
    [ O ,  X ,  - ]
    [ X ,  O ,  - ]

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

O - Select row (1-3) & select column (1-3) separated by a space: 2 3
```

```
    [ X ,  O ,  X ]
    [ O ,  X ,  O ]
    [ X ,  O ,  - ]

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

REFLECT

As noted, this version of the program handles all of the turns. In addition to thinking about the common elements in each turn, what do you think the next steps in the development process will be?

You have now replaced the paper version of tic-tac-toe with a computerized version. However, you may notice that there's a lot of repeating elements. It is missing other features, including things like checking if a player has already won. Additionally, it is missing a check to see if the player is entering a value that is within the correct range. During future tutorials on loops, you will learn how we can trim this program down and optimize the code even further.

SUMMARY

You **reviewed the use of arrays**. This included the use of **one-dimensional arrays**. Based on the properties of the data, you learned that a **two-dimensional array** was the best match for our **Tic-Tac-Toe program**, since it was known that the number of elements is fixed, but the values in the elements will need to change. You learned that the order of the elements is important to represent the state of the game. You designed and implemented the board of the program using nested lists and structured it to resemble a real-world Tic-Tac-Toe board. Finally, you programmed both players' input and sent the results to the screen. This program does not handle incorrect player input; the player chooses an element that is already taken. This will be covered in future tutorials to optimize the code.

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/