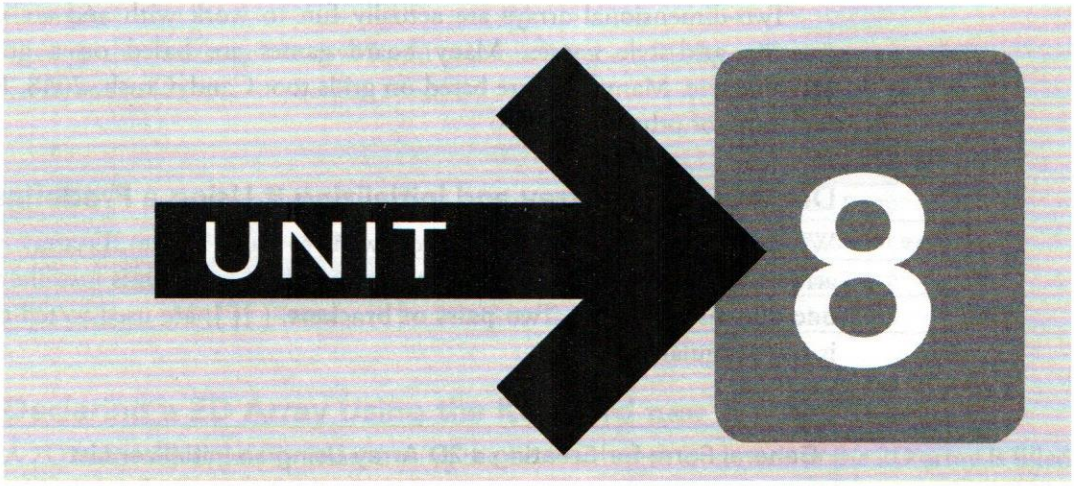


2D Array

IN THIS UNIT

Summary: The data structures that we've worked with so far have all been one-dimensional arrays because they stored a simple list of data values. Sometimes using an array or `ArrayList` is not enough. A two-dimensional array stores values in rows and columns like in a table. In this unit we will discuss 2D arrays. We will also look back at some of the code that we already wrote and modify it to work with a 2D array.

Key Ideas

- A two-dimensional (2D) array is a data structure that is an array of arrays.
- The 2D array simulates a rectangular grid with coordinates for each location based on the row and the column.
- 2D arrays can be traversed using row-major or column-major order.
- All standard array or `ArrayList` algorithms can be applied to 2D arrays.

The 2D Array

Definition of a 2D Array

A **two-dimensional (2D) array** is a complex data structure that can be visualized as a rectangular **grid** made up of **rows** and **columns**. Technically, it is **an array of arrays**. It can store any kind of data in its slots; however, each piece of data has to be of the same data type.

Two-dimensional arrays are actually fun to work with and are very practical when creating grid-style games. Many board games are based on a grid: checkers, chess, Scrabble, etc. Many apps are based on grids too: CandyCrush, 2048, Ruzzle. I'm sure you can think of others.

Declaring a 2D Array and Initializing It Using a Predefined List of Data

When you know the values that you want to store in the 2D array, you can declare the array and store the values in it immediately. This operation is just like what we did for the one-dimensional array. **Two pairs of brackets**, [ ][ ], are used to tell the computer that it is not a regular variable.

General Form for Creating a 2D Array Using an Initializer List

```
dataType[][] nameOf2DArray = { {value1, value2, value3},
                               {value4, value5, value6},
                               {. . . , . . . , . . . } };
```

value1	value2	value3
value4	value5	value6
. . .	. . .	. . .

Note: Pairs of curly braces are used to wrap the first row, then the second row, and so on. If you look closely at this visual, you will see how a 2D array is really an array of arrays.

Example

Declare a 2D array that represents a CandyCrush board with four rows and three columns. Notice that the four rows are numbered 0 through 3 and the three columns are numbered 0 through 2.

```
String[][] candyBoard = {{"Jelly Bean", "Lozenge", "Lemon Drop"},
                         {"Gum Square", "Lollipop Head", "Jujube Cluster"},
                         {"Lozenge", "Lollipop Head", "Lemon Drop"},
                         {"Jelly Bean", "Lollipop Head", "Lozenge"}};
```

This is the visual representation of what the candyBoard array looks like inside the computer.

	0	1	2
0	"Jelly Bean"	"Lozenge"	"Lemon Drop"
1	"Gum Square"	"Lollipop Head"	"Jujube Cluster"
2	"Lozenge"	"Lollipop Head"	"Lemon Drop"
3	"Jelly Bean"	"Lollipop Head"	"Lozenge"

Rows and Columns of the 2D Array

Every cell in a 2D array is assigned a pair of coordinates that are based on the row number and the column number. The first pair of brackets represents the row number and the second pair of brackets represents the column number. The rows and columns both begin at zero and end with one less than the number of rows or columns, just like they did with arrays and ArrayLists.

Example

Change the "Lozenge" in row 2 and column 0 to be a "Lemon Drop".

```
candyBoard[2][0] = "Lemon Drop";
```

	0	1	2
0	"Jelly Bean"	"Lozenge"	"Lemon Drop"
1	"Gum Square"	"Lollipop Head"	"Jujube Cluster"
2	"Lemon Drop"	"Lollipop Head"	"Lemon Drop"
3	"Jelly Bean"	"Lollipop Head"	"Lozenge"

Declaring a 2D Array Using the Keyword new

A 2D array object can be created using the keyword **new**. Every cell in the 2D array is filled with the default value for its data type.

General Form for Creating a 2D Array Using the Keyword new

```
dataType[][] nameOfArray = new dataType[numberOfRows][numberOfColumns];
```

Example

Declare a 2D array that represents a Sudoku Board that has nine rows and nine columns. Put the number 6 in row 2, column 7:

```
int[][] mySudokoBoard = new int[9][9]; // the order is always [rows][columns]
mySudokoBoard[2][7] =  6;              // puts a 6 in row 2, column 7
```

9 Columns

9 Rows

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	6	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0

How to Refer to the Rows and Columns in a 2D Array

The position, `myBoard[0][5]`, is read as *row 0, column 5.*

The position, `myBoard[3][0]`, is read as *row 3, column 0.*

Using the `length` Field to Find the Number of Rows and Columns

The number of rows in a 2D array is found by accessing the `length` field. The number of columns in a 2D array is found by accessing the `length` field on the name of the array *along with one of the rows* (it doesn't matter which row).

Example 1

Retrieve the number of rows from a 2D array:

```
double[][] myBoard = new double[8][3];
int result = myBoard.length;        // result is 8 (number of rows)
```

Example 2

Retrieve the number of columns from a 2D array:

```
double[][] myBoard = new double[8][3];
int result1 = myBoard[0].length;    // result1 is 3 (number of columns)
int result2 = myBoard[5].length;    // result2 is also 3
```

Accessing a 1-D Array from Within the 2D Array

Consider this 2D array declaration:

```
double[][] myBoard = new double[8][3];
```

`myBoard[0]` is a 1-D array that consists of the row with index 0.
`myBoard[5]` is a 1-D array that consists of the row with index 5.

Traversing a 2D Array in Row-Major Order

Traversing a 2D array means to visit every cell in the grid. This can be done in many different ways, but for the AP Computer Science A exam, you need to know two specific ways.

Row-Major order is the process of traversing a 2D array in the manner that English-speaking people read a book. Start at the top left cell and move toward the right along the first row until you reach the end of the row. Then start at the next row at the left-most cell, and then move along to the right until you reach the end of that row. Repeat this process until you have visited every cell in the grid and finish with the bottom-right cell.

Example

Traverse an array in Row-Major order. Start with row 0 and end with the last row. For each row, start with column 0 and end with the last column.

Java Code 1: Using a nested for loop

```
String[][] myGrid =  /* 2D array provided by the user */
for (int row = 0; row < myGrid.length; row++)                    // every row
{
   for (int column = 0; column < myGrid[0].length; column++)     // every column
   {
     System.out.print(myGrid[row][column] + " ");                // every cell
   }
   System.out.println( );                          // so it prints like a grid
}
```

Java Code 2: Using enhanced for loop

```
String[][] myGrid =    /* 2D array provided by the user */
for (String [] arr : myGrid)          // put each row into an array
{
    for (String element : arr)          // traverse one single row
    {
        System.out.print(element + " ");     // one cell
    }
    System.out.println( );                // so it prints like a grid
}
```

Traversing a 2D Array in Column-Major Order

Column-Major order is the process of traversing a 2D array by starting at the top-left cell and moving downward until you reach the bottom of the first column. Then start at the top of the next column and work your way down until you reach the bottom of that column. Repeat this until you have visited every cell in the grid and finish with the bottom-right cell.

Example

Traverse an array in column-major order using a nested `for` loop. Start with column 0 and end with the last column. For each column, start with row 0 and end with the last row.

```
String[][] myGrid =  /* 2D array provided by the user */
for (int column = 0; column < myGrid[0].length; column++) // every column
{
   for (int row = 0; row < myGrid.length; row++)          // every row
   {
      System.out.println(myGrid[row][column]);           // every cell
   }
}
```

`ArrayIndexOutOfBoundsException`

As with a 1-D array, if you use an index that is not within the 2D array, you will get an `ArrayIndexOutOfBoundsException`.

Fun Fact: *Java provides support for multi-dimensional arrays, such as 3D arrays; however, the AP Computer Science A Exam does not require you to know about them.*

More Algorithms

All standard array/`ArrayList` algorithms can be applied to 2D arrays. Let's revisit a few of them from units 6 and 7.

The Accumulate Algorithm

Remember the baseball scenario from unit 6 where we found the total number of hits the team had? We are going to make modifications so instead of finding the total hits from one game (1D array) we will find the total number of hits for all the games in the season (2D array).

General Problem: Add up all of the hits for a baseball team.

Refined Problem: Write a method that finds the sum of all of the hits in an 2D array of hits.

Final Problem: Write a method called `findSum` that has one parameter: a 2D array of ints. The method should return the sum of all of the elements in the array.

Algorithm:
Step 1: Create a variable called sum and set it equal to zero
Step 2: Look at each of the elements in the list
Step 3: Add the element to the sum
Step 4: Continue until you reach the end of the list
Step 5: Return the sum

Pseudocode:

```
set sum = zero
for (iterate through all the elements in the list)
{
   sum = sum + element
}
return sum
```

Java code using a 2D array (Row-Major order)

```
public static int findSum(int[][] arr)
{
   int sum = 0;                                    // initialize sum to 0
   for (int row = 0; row < arr.length; row++)      // do every row
   {
      for (int col = 0; col < arr[row].length; col++)
      {                                            // do every column in the row
         sum += arr[row][col];                     // add value in cell to sum
      }
   }
   return sum;                                     // return sum
}
```

The Find-Highest Algorithm

What is the highest score on your favorite video game? As more people play a game, how does the computer figure out what is the highest score in the list?

General Problem: Find the highest score out of all the scores for a video game.

Refined Problem: Write a method that finds the highest score in a list of scores.

Final Problem: Write a method called `findHigh` that has one parameter: a 2D array. The method should return the largest value in the array.

Solution: Let the highest be the first element in the list.

Algorithm:
Step 1: Create a variable called high and set it to the first element in the list
Step 2: Look at each of the scores in the list
Step 3: If the score is greater than the high score, then make it be the new high score
Step 4: Continue until you reach the end of the list
Step 5: Return the high score

Pseudocode:

```
set high = first score
for (iterate through all the scores in the list)
{
  if (score > high)
  {
    high = score
  }
}
return high
```

Java code using a 2D array (Row-Major order)

```
public static int findHigh(int[][] arr)
{
    int high = arr[0][0];      // first element
    for (int row = 0; row < arr.length; row++)
    {
        for (int col = 0; col < arr[row].length; col++)
        {
            if (arr[row][col] > high)
            {
                high = arr[row][col];
            }
        }
    }
    return high;
}
```

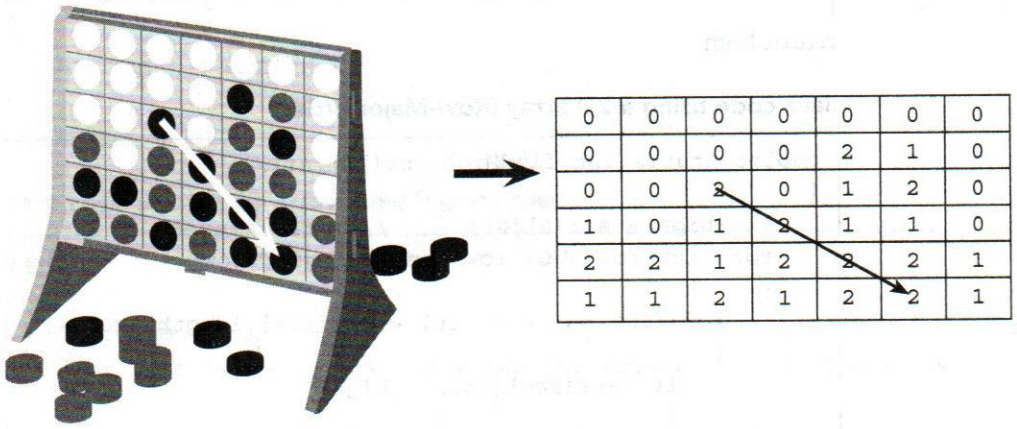
The Connect-Four Advanced Algorithm

Connect Four is a two-player game played on a grid of six rows and seven columns. Players take turns inserting disks into the grid in hopes of connecting four disks in a row to win the game. You have been hired to work on the app version of the game, and it is your responsibility to determine whether a player has won the game.

General Problem: Determine if a player has won the game of Connect Four.

Refined Problem: The Connect Four app version uses a 2D array that is 6 × 7. After a player makes a move, determine if he or she has won by checking all possible ways to win. If four of the same color disks are found in a line, then the game is over. Decide who won the game (player one or player two). A way to win can be vertical, horizontal, diagonally downward, or diagonally upward.

Final Problem: Write a method called `checkForWinner` that has one parameter: a 2D array of `int`. The 2D array consists of 1s and 2s to represent a move by player one or player two. Cells that do not contain a move have a value of 0. This method checks all four directions to win (vertical, horizontal, diagonally downward, and diagonally upward). The method should return the number of the player who won or return 0 if nobody has won.



0	0	0	0	0	0	0
0	0	0	0	2	1	0
0	0	2	0	1	2	0
1	0	1	2	1	1	0
2	2	1	2	2	2	1
1	1	2	1	2	2	1

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	2	1	0
2	0	0	2	0	1	2	0
3	↓	↓	↓	↓	1	1	0
4	2	2	1	2	2	2	1
5	1	1	2	1	2	2	1

Algorithm (for vertical win):
Note: There are a total of 21 ways to win vertically. Start with the top left cell and move in Row-Major order.

Step 1: Start with row 0 and end with row 2 (row 2 is four less than the number of rows)
Step 2: Start with column 0 and end with column 6 (use every column)
Step 3: If current cell value is not equal to 0 and the current cell = cell below it = cell below it = cell below it, then a player has won
Step 4: Continue until you reach the end of row 2
Step 5: Return the player who won or zero

Pseudocode (for vertical win):

```
for (start with row 0 and end with row 2)
{
   for (start with column 0 and end with column 6)
   {
      if (current cell != 0 && current cell = cell below it = cell below it = cell below it)
      {
         return current cell value
      }
   }
}
return 0 (no winning player is found)
```

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	2	1	0
2	0	0	2	0	1	2	0
3	1	0	1	2	1	1	0
4	2	2	1	2	2	2	1
5	1	1	2	1	2	2	1

Algorithm (for diagonal downward win):

Note: There are a total of 12 ways to win diagonally downward. Start with the top left cell and go in Row-Major order.

Step 1: Start with row 0 and end with row 2 (row 2 is four less than the number of rows)
Step 2: Start with column 0 and end with column 3 (column 3 is four less than the number of columns)
Step 3: If current cell value is not equal to 0 and the cell below and to the right = cell below and to the right = cell below and to the right, then a player has won
Step 4: Continue until you reach the end of row 2
Step 5: Return the player who won or zero

Pseudocode (for diagonal downward win):

```
for (start with row 0 and end with row 2)
{
  for (start with column 0 and end with column 3)
  {
    if (current cell != 0 && current cell = cell below/right = cell below/right
       = cell below/right)
    {
       return current cell value
    }
  }
}
return 0
```

Note: I have provided the algorithm and pseudocode for only the vertical win and the diagonal downward win. The code for these, as well as the horizontal win and the diagonal upward win, are provided in the Java code that follows.

Java Code: Using a 2D array and nested `for` loops (includes all four ways to win):

```
public static int checkForWinner(int[][] board)
{
   // Check for Vertical Win
   for (int r = 0; r <= board.length - 4; r++)
   {
      for (int c = 0; c < board[0].length; c++)
      {
         if (board[r][c] != 0 &&
             board[r][c] == board[r+1][c] &&
             board[r+1][c] == board[r+2][c] &&
             board[r+2][c]  == board[r+3][c])
         return board[r][c];
      }
   }

   // Check for Diagonal Downward Win
   for (int r = 0; r <= board.length - 4; r++)
   {
      for (int c = 0; c <= board[0].length - 4; c++)
      {
         if (board[r][c] != 0 &&
             board[r][c] == board[r+1][c+1] &&
             board[r+1][c+1] == board[r+2][c+2] &&
             board[r+2][c+2] == board[r+3][c+3])
         return board[r][c];
      }
   }

   // Check for Horizontal Win
   for (int r = 0; r < board.length; r++)
   {
      for (int c = 0; c <= board[0].length - 4; c++)
      {
         if (board[r][c] != 0 &&
             board[r][c] == board[r][c+1] &&
             board[r][c+1] == board[r][c+2] &&
             board[r][c+2] == board[r][c+3])
         return board[r][c];
      }
   }

   // Check for Diagonal Upward Win
   for (int r = 3; r < board.length; r++)
   {
      for (int c = 0; c <= board[0].length - 4; c++)
      {
         if (board[r][c] != 0 &&
             board[r][c] == board[r-1][c+1] &&
             board[r-1][c+1] == board[r-2][c+2] &&
             board[r-2][c+2] == board[r-3][c+3])
```

```
            return board[r][c];
        }
    }

    // No winner was found after checking all 4 ways to win
    return 0;
}
```

```
public class ConnectFourRunner
{
    public static void main(String[] args)
    {
        int[][] board = {{0,0,0,0,0,0,0},
                         {0,0,0,0,2,1,0},
                         {0,0,2,0,1,2,0},
                         {1,0,1,2,1,1,0},
                         {2,2,1,2,2,2,1},
                         {1,1,2,1,2,2,1} };

        System.out.println("Winner:  " + checkForWinner(board));
    }

    public static int checkForWinner(int[][] arr)
    {
        /*  implementation is described above  */
    }
}
```

```
OUTPUT
Winner:  2
```

› Rapid Review

The 2D Array

- A 2D array is a complex data structure that can store a grid of data of the same type.
- The 2D array is actually an array of arrays.
- The rows and columns are numbered starting with zero.
- The top row is row zero. The left-most column is column zero.
- The access to each cell in the 2D array is always [*row*][*column*].
- The indices of the top-left cell are [0][0].
- The index of the bottom-right cell is [*numberOfRows* − 1][*numberOfColumns* − 1].
- Two-dimensional arrays can store primitive data or object data.
- To store a value in a 2D array: arrayName[*rowNumber*][*columnNumber*] = value;
- To retrieve the number of rows in a 2D array, use arrayName.length.
- To retrieve the number of columns in a rectangular 2D array, use arrayName[0].length. Note: Any valid row number works instead of 0.
- Using an index that is not in the range of the 2D array will throw an `ArrayIndexOutOfBoundsException`.

Traversing a 2D Array

- To traverse a 2D array means to visit each element in every row and column.
- The most common way to traverse a 2D array is called Row-Major order.
- Row-Major order starts in the upper-left location [0][0], and travels to the right until the end of the row is reached. Then the next move is to the next row [1][0]. The process is repeated until the right-most column in the bottom row is reached.
- The second most common way to traverse a 2D array is called Column-Major order.
- Column-Major order also starts in the upper-left location [0][0]; however, it travels down until the end of the first column is reached. Then, the next move is to the top of the next column [0][1]. The process is repeated until the bottom row in the right-most column is reached.
- All of the 2D arrays on the AP Computer Science A Exam will be rectangular.

Accumulate Algorithm

- The accumulate algorithm finds the sum of all the items in a list.
- It looks at each element in the list one at a time, adding the value to a total.

Find-Highest Algorithm

- The find-highest algorithm finds the largest value in a list.
- It looks at each element in the list one at a time, comparing the value to the current high value.
- Common ways to implement this algorithm include initializing the highest value with the first value in the list or to an extremely small negative number.
- The find-highest algorithm can be modified to find the lowest item in a list.

› Review Questions

Basic Level

1. Consider the following code segment.

```
int n = // some positive integer
int [ ][ ] table = new int[n][n];
//  input values into table

int sum = 0;
for (int row = 0; row < table.length; row++)
     sum += table[table.length - row - 1][row];

System.out.println(sum);
```

What value will `sum` contain after the code segment is executed?

(A) The sum of all the values in the table
(B) The sum of all the values in the major diagonal (top left to bottom right)
(C) The sum of all the values in the minor diagonal (top right to bottom left)
(D) The sum of all the values in both diagonals
(E) The sum of all the values in the last row

2. Consider the following code segment.

```
int m = // some positive integer
int n = // some positive integer
int [ ][ ] table = new int[m][n];
//  input values into table

for (int row = 0; row < table.length; row++)
    for (int col = 0; col < table[0].length; col++)
        if(table[row][col] < 0)
            table[row][col] = - table[row][col];
```

Which of the following best describes the result of executing the code segment?

(A) Each element in the two-dimensional array `table` contains the value 0.
(B) Each element in the two-dimensional array `table` contains a nonnegative value.
(C) Each element in the two-dimensional array `table` contains a nonpositive value.
(D) Each element in the two-dimensional array `table` contains the opposite value from when it was initialized.
(E) Each element in the two-dimensional array `table` contains the value row – col.

3. Consider the following incomplete method.

```
public static double findRowSum(double [][] mat, int n)
{
    double sum = 0;
    // missing code

    return sum;
}
```

Method `findRowSum` is intended to return the sum of elements in parameter `mat` that are in row n, also passed as a parameter. Which of the following code segments could be used to replace // missing code so that `findRowSum` will work as intended?

(A)
```
for (int i = 0; i < table.length; i++)
    sum += mat[i][n];
```

(B)
```
for (int i = 0; i < table[0].length; i++)
    sum += mat[i][n];
```

(C)
```
for (int i = 0; i < table.length; i++)
    sum += mat[n][i];
```

(D)
```
for (int i = 0; i < table[0].length; i++)
    sum += mat[n][i];
```

(E)
```
for (int i = 0; i < table[0].length; i++)
    sum += mat[n][n];
```

Advanced Level

4. Consider the following definition and code segment.

```
int [][] table =  new int[5][5];

for (int row = 0; row < table.length-1; row++)
    for (int col = 0; col < table[0].length-1; col++)
    {
            table[row][col] = row * col;
    }
```

What values will `table` contain after the code segment is executed?

(A)

0	0	0	0	0
0	1	2	3	4
0	2	4	6	8
0	3	6	9	12
0	4	8	12	16

(B)

0	1	2	3	4
1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8

(C)

0	1	2	3	0
1	2	3	4	0
2	3	4	5	0
3	4	5	6	0
0	0	0	0	0

(D)

0	0	0	0	0
0	1	2	3	0
0	2	4	6	0
0	3	6	9	0
0	0	0	0	0

(E)

0	0	0	0	0
0	3	6	9	0
0	2	4	6	0
0	1	2	3	0
0	0	0	0	0

5. Modifying a 2D array

Write a method that takes a 2D array as a parameter and returns a new 2D array.
The even rows of the new array should be exactly the same as the array passed in.
The odd rows of the new array should be replaced by the contents of the row above.

For example, if the input array is:

1	5	4	8	7	3
2	4	3	5	7	6
2	4	3	3	6	0
9	8	9	9	4	1

Then the returned array should be:

1	5	4	8	7	3
1	5	4	8	7	3
2	4	3	3	6	0
2	4	3	3	6	0

Here is the declaration for your method.

```
public int[][] modify(int[][] arr)
```

6. Filling a 2D array

Write a method that will fill in a 2D `boolean` array with a checkerboard pattern of alternating `true` and `false`. The upper-left corner (0, 0) should always be `true`.

Given a grid size of 3 × 4, the method should return:

true	false	true	false
false	true	false	true
true	false	true	false

The method will take the number of rows and columns as parameters and return the completed array. Here is the method declaration and the array declaration:

```
public static boolean[][] makeGrid(int rows, int cols)
{
   boolean[][] grid;

   /* to be implemented */
}
```

7. Find all the songs that contain the word "Love" in the title.

One of the most popular themes in music lyrics is love. Suppose a compilation of thousands of songs is stored in a grid. Count the number of songs that contain the word "Love" in the title.

Write a method that counts the number of song titles that contains a certain string in the title. The method has two parameters: a 2D array of `String` objects and a target string. The method should return an integer that represents the number of strings in the 2D array that contains the target string.

```
String[][] songs = /*  2D array fill with the values in the table below  */
int result = findCount(songs, "Love");                    // result is 5
```

The 2D array passed to the method:

"We Are the Champions"	"You Shook Me All Night Long"	"We Found Love"
"Bleeding Love"	"Stairway to Heaven"	"Won't Get Fooled Again"
"I'd Do Anything for Love"	"Stupid Crazy Love"	"Love in This Club"
"Since U Been Gone"	"One More Time"	"Walk This Way"

```
public int findCount(String[][] arr, String target)
{
   // Write the implementation
}
```