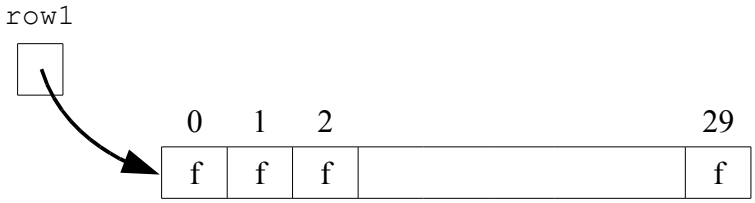
Arrays in Java – Multi-dimensional Arrays

Suppose you are tasked with writing a program to help maintain seating records for a theatre company. The auditorium has 25 rows, each of which contains 30 seats. One utility you need to provide is tracking which seats are sold, which can be accomplished using boolean values.

We could use a single array for each row, which would require the following declaration:

```
boolean[] row1 = new boolean[30];
boolean[] row2 = new boolean[30];
...
boolean[] row25 = new boolean[30];
```

Each row could be visualized as

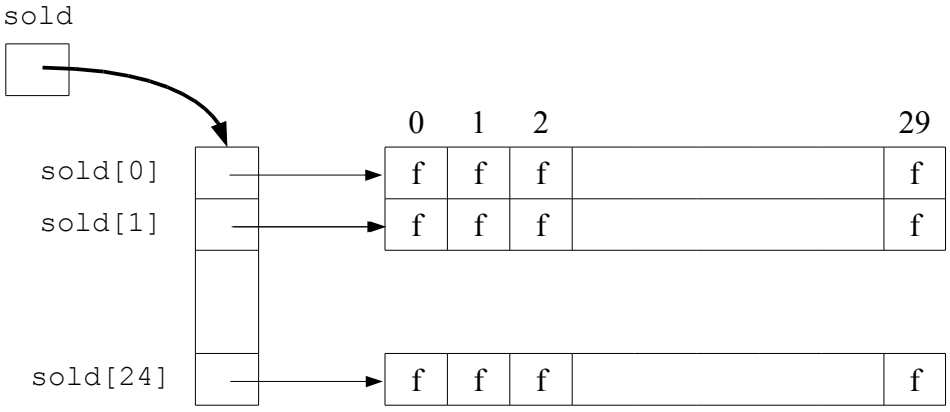


Obviously this approach requires too much manual declaration of each row, and the program becomes difficult to maintain even if simple changes are made to the number of rows or the number of seats.

Fortunately, Java allows the declaration of multi-dimensional arrays.

```
boolean[][] sold = new boolean[25][30];
```

The results of the declaration are shown the following diagram:



This diagram may somewhat surprising, but it is consistent with our previous considerations of a one-dimensional array. If we begin with the declaration

```
boolean[] sold = new boolean[25];
  (3)   (2) (1)
```

we are declaring a reference (1) to an array object (2) that contains boolean elements (3).

Now consider the declaration of our two-dimensional array.

```
boolean[][] sold = new boolean[25][30];
   (3)   (2) (1)
```

By adding the second set of square brackets, we have declared a reference variable (1) to an array object (2) that contains *references to arrays of boolean elements*. Specifically, we have 25 arrays of 30 boolean elements.

Often, we use two-dimensional arrays to represent values in a simple rectangular table. In such cases, we can think of the two-dimensional array as consisting of rows (horizontal) and columns (vertical), like a spreadsheet. If we do this, it is customary to think of the row index coming before the column index.

```
boolean[][] sold = new boolean[rows][columns];
```

Traversing a Two-dimensional Array

Just as the for statement is useful for traversing a one-dimensional array, we can use a *nested for* statement to traverse a two-dimensional array.

```
int[][] table = new int[3][6];
```

will create an array with 3 rows and 6 columns, and by default each element will automatically be assigned the value of zero.

		columns					
		0	1	2	3	4	5
	0	0	0	0	0	0	0
rows	1	0	0	0	0	0	0
	2	0	0	0	0	0	0

If we wished to initialize the array elements to one, we could use the following code fragment:

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

Initializing a Two-dimensional Array at Declaration

Suppose we want to create a table with 3 rows and 4 columns, initialized as follows:

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

We can do so with the following statement:

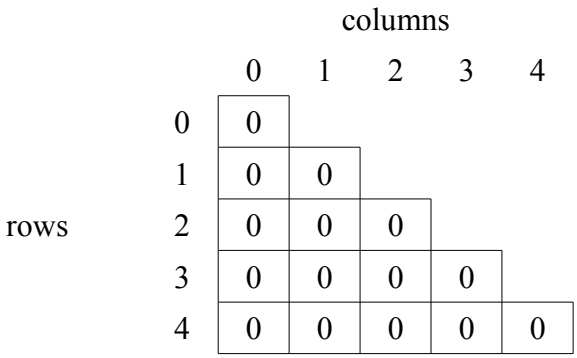
```
int[][] scores = {{5,2,7,3},
                  {6,8,7,9},
                  {4,5,2,3}};
```

Notice that we have used multiple lines to represent the statement. This is strictly for readability. The equivalent statement, on a single line, would work equally well.

```
int[][] scores = {{5,2,7,3}, {6,8,7,9}, {4,5,2,3}};
```

Non-rectangular Two-dimensional Arrays

The one-dimensional arrays that compose a two-dimensional array do not need to be the same length, although this is certainly the most common configuration. Consider the following triangular array:



In our declaration of such an array, we cannot specify the number of columns for each, since that size changes with each row. Thus we leave the number of columns blank for now.

```
double[][] triTable = new double[5][];
```

Now we must dynamically create the array for each row to a custom size, which can be accomplished with the following code fragment:

```
for (int row = 0; row < triTable.length; row++)
{
     triTable[row] = new double[row+1];
}
```

Note: Array declarations such as the one above, where the size of the second dimension is left blank, are legal. However, it is *not* legal to have the first dimension unspecified.

```
int[][] badArray = new int[][20];  // generates an error, will not compile
```

Non-rectangular two-dimensional arrays are sometimes called *ragged arrays*.

Initializing a Ragged Array

```
int[][] ragged = {{4, 3, 7},
                  {5, 2},
                  {7, 8, 1, 4}};
```

creates the array whose elements are illustrated below.

4	3	7	
5	2		
7	8	1	4

Traversing a Ragged Array

In traversing a ragged two-dimensional array, we cannot assume that each row contains the same numbers of columns. As such, our nested for loops must be slightly modified.

Suppose we want to find the sum of the elements in the ragged array declared above.

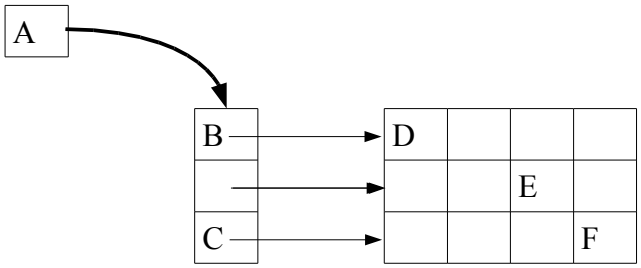
```
int sum = 0;
for (int row = 0; row < ragged.length; row++)
{
      for (int col = 0; col < ragged[row].length; col++)
      {
            sum += ragged[row][col];
      }
}
```

Exercises

1. The diagram shows an array declared by the statement

```
int[][] a = new int[3][4];
```

State the identifier of each cell marked by a letter.



2. How many elements would be in each of the arrays created by the following declarations?

a) `double[][] first = new double[25][40];`

b) `boolean [][][] second = new boolean[3][6][50];`

c) `char [][] third = new char[60][40];`

3. Suppose that the following declarations have been made:

```
int a[][] = {{4,2,7},
             {3,9,1}};
```

Determine what would be printed by each fragment

a)
```
for (i = 0; i < a.length; i++)
{
      for (j = 0; j < a[0].length; j++)
      {
            System.out.print(a[i][j]);
      }
      System.out.println();
}
```

b)
```
for (i = 0; i < a[0].length; i++)
{
      for (j = 0; j < a.length; j++)
      {
            System.out.print(a[j][i]);
      }
      System.out.println();
}
```

c)
```
for (i = a.length - 1; i >= 0; i--)
{
      for (j = 0; j < a[0].length; j++)
      {
            System.out.print(a[i][j]);
      }
      System.out.println();
}
```

4. For the array given in the previous question, write a fragment that would print the elements of the array in the form

   17
   92
   34

5. Write a method `sum` having one `double[][]` parameter. The method should return the sum of the elements of the array passed to it. You may assume that the array is rectangular.

6. Write a method `max` that will return the maximum value of the elements in a two-dimensional array of int values. Do *not* assume that the array is rectangular.

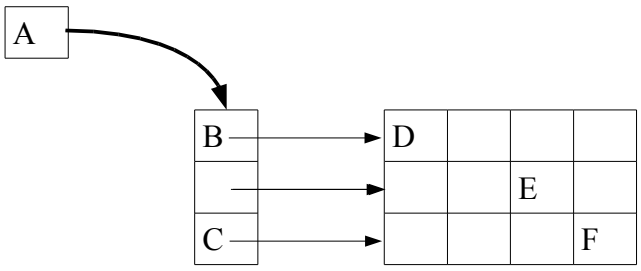
7. Write a method `size` that has one `int[][][]` parameter. The method should return the number of elements in the array. Do not make any assumptions about regularity of the array.

Solutions

1. The diagram shows an array declared by the statement

```
int[][] a = new int[3][4];
```

State the identifier of each cell marked by a letter.



A) a
B) a[0]
C) a[2]
D) a[0][0]
E) a[1][2]
F) a[2][3]

2. How many elements would be in each of the arrays created by the following declarations?

a) `double[][] first = new double[25][40];`
1000 elements

b) `boolean [][][] second = new boolean[3][6][50];`
900 elements

c) `char [][] third = new char[60][40];`
2400 elements

3. Suppose that the following declarations have been made:

```
int a[][] = {{4,2,7},
             {3,9,1}};
```

Determine what would be printed by each fragment

a)
```
for (i = 0; i < a.length; i++)
{
      for (j = 0; j < a[0].length; j++)
      {
            System.out.print(a[i][j]);
      }
      System.out.println();
}
```

Output

427
391

b)
```
for (i = 0; i < a[0].length; i++)
{
      for (j = 0; j < a.length; j++)
      {
            System.out.print(a[j][i]);
      }
      System.out.println();
}
```

Output

```
43
29
71
```

c)
```
for (i = a.length - 1; i >= 0; i--)
{
      for (j = 0; j < a[0].length; j++)
      {
            System.out.print(a[i][j]);
      }
      System.out.println();
}
```

Output

```
391
427
```

4. For the array given in the previous question, write a fragment that would print the elements of the array in the form

```
17
92
34
```

```
for (i = a[0].length - 1; i >= 0; i--)
{
      for (j = a.length - 1; j >= 0; j--)
      {
            System.out.print(a[j][i]);
      }
      System.out.println();
}
```

5. Write a method `sum` having one `double[][]` parameter. The method should return the sum of the elements of the array passed to it. You may assume that the array is rectangular.

```
public static double sum (double[][] array)
{
  double sum = 0;
  for (int i = 0; i < array.length; i++)
  {
    for (int j = 0; j < array[0].length; j++)
    {
      sum += array[i][j];
    }
  }
  return sum;
}
```

6. Write a method `max` that will return the maximum value of the elements in a two-dimensional array of int values. Do *not* assume that the array is rectangular.

```
public static int max ( int[][] array)
{
  int max = array[0][0];  // need a starting value
  for (int i = 0; i < array.length; i++)
  {
    for (int j = 0; j < array[i].length; j++)
    {
      if (array[i][j] > max)
        max = array[i][j];
    }
  }
  return max;
}
```

7. Write a method `size` that has one `int[][][]` parameter. The method should return the number of elements in the array. Do not make any assumptions about regularity of the array.

```
public static int size ( int[][][] array)
{
  int size = 0;  // need a starting value
  for (int i = 0; i < array.length; i++)
  {
    for (int j = 0; j < array[i].length; j++)
    {
      for (int k = 0; k < array[i][j].length; k++)
        size++;
    }
  }
  return size;
}
```