

Arrays

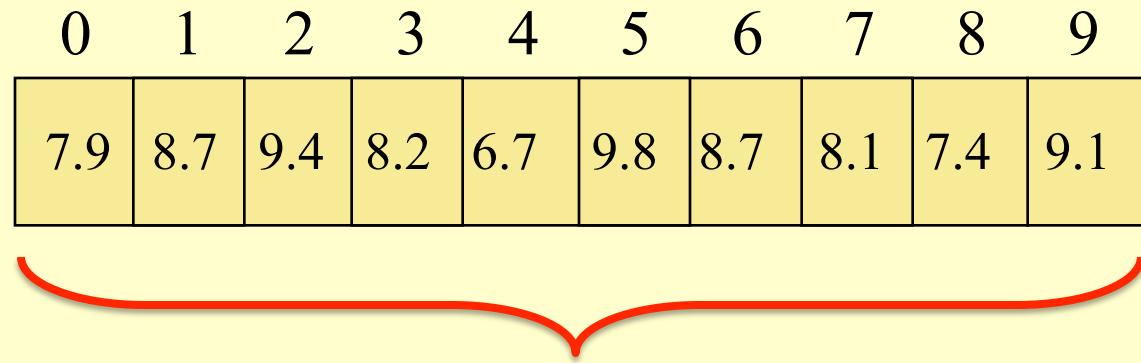
CSC 2014 – Java Bootcamp

Dr. Mary-Angela Papalaskari
Department of Computing Sciences
Villanova University

Some slides in this presentation are adapted from the slides accompanying Java Software Solutions by Lewis & Loftus

Arrays

- An array is an ordered list of values:



This array holds 10 values of type `double` that are indexed from 0 to 9

Arrays - Overview

- Declaration:

```
double[] scores = new double[10];
```

element type

- Instantiation:

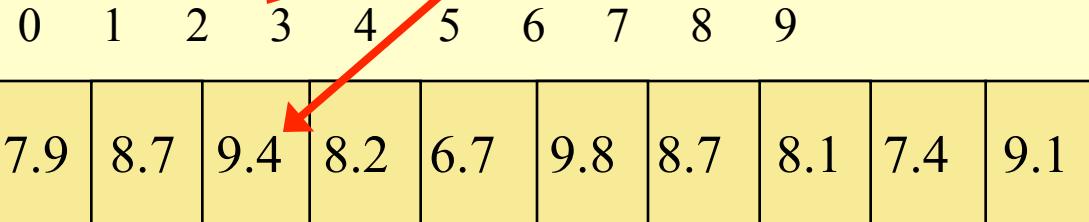
The entire array has a single name

scores

index

array element

scores[2]



- Initialization:

```
scores[0] = 7.9;  
scores[1] = 8.7;  
scores[2] = 9.4;  
scores[3] = 8.2;  
scores[4] = 6.7;  
scores[5] = 9.8;  
scores[6] = 8.7;  
scores[7] = 8.1;  
scores[8] = 7.4;  
scores[9] = 9.1;
```

- This array holds 10 values of type `double` that are indexed from 0 to 9
- The size of the array is given by: `scores.length = 10`

- Alternative declaration/instantiation/Initialization:

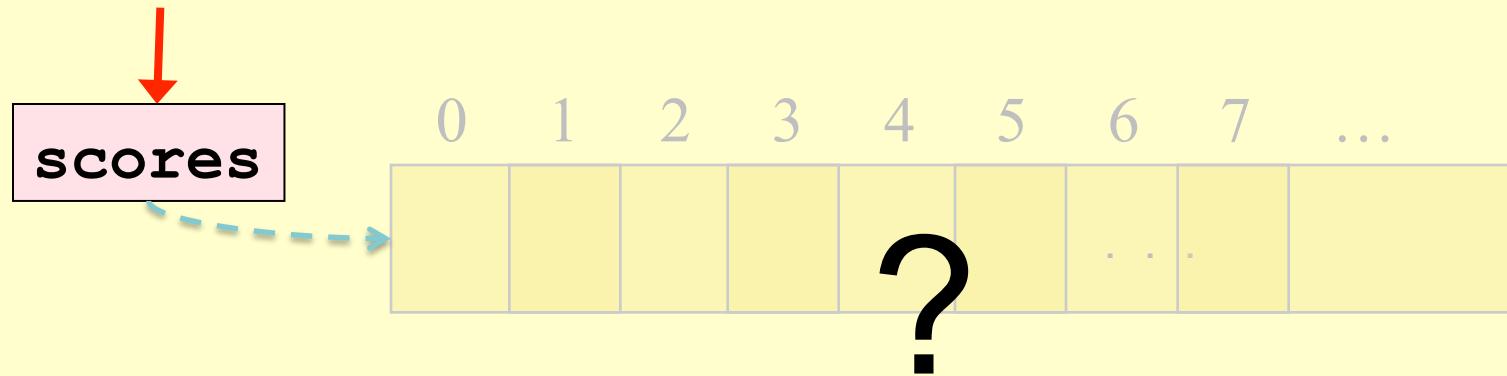
```
double[] scores = {7.9, 8.7, 9.4, 8.2, 6.7, 9.8, 8.7, 8.1, 7.4, 9.1};
```

Arrays - Declaration

element type

- Declaration: `double[] scores`

The entire array
has a single name



Arrays - Instantiation

- Declaration:

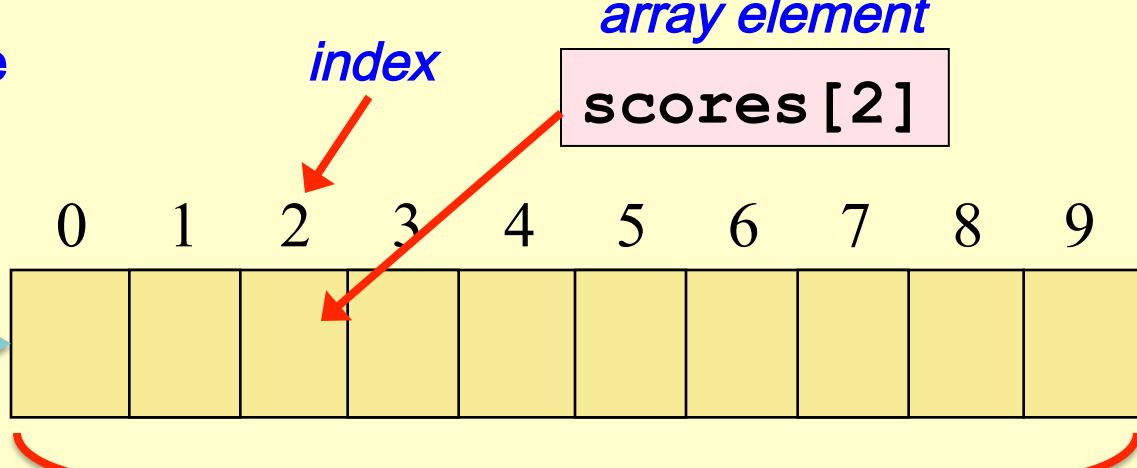
```
double[] scores = new double[10];
```

element type

• Instantiation:

The entire array
has a single name

scores



Size of array

`scores.length`

10

Arrays - Initialization

- Declaration:

```
double[] scores = new double[10];
```

element type

- Instantiation:

The entire array has a single name

scores

index

array element

scores[2]

0	1	2	3	4	5	6	7	8	9
7.9	8.7	9.4	8.2	6.7	9.8	8.7	8.1	7.4	9.1

- Initialization:

```
scores[0] = 7.9;  
scores[1] = 8.7;  
scores[2] = 9.4;  
scores[3] = 8.2;  
scores[4] = 6.7;  
scores[5] = 9.8;  
scores[6] = 8.7;  
scores[7] = 8.1;  
scores[8] = 7.4;  
scores[9] = 9.1;
```

Size of array

scores.length

10

Declaring and instantiating Arrays

- Some more examples:

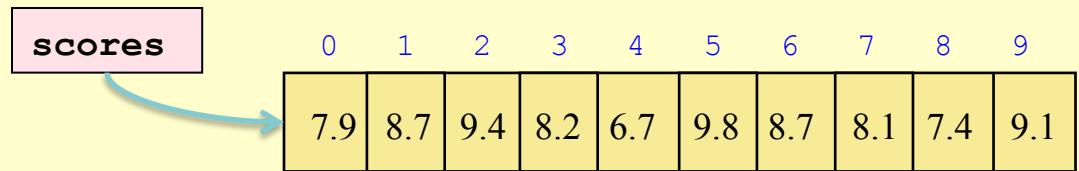
```
int[] weights = new int[2000];
```

```
boolean[] flags;  
flags = new boolean[20];
```

```
char[] codes = new char[1750];
```

```
double[] prices = new double[500];
```

Using Arrays



Array elements can be assigned a value, printed, or used in a calculation. Examples:

```
System.out.println ("Top = " + scores[5]);
```

```
mean = (scores[0] + scores[1])/2;
```

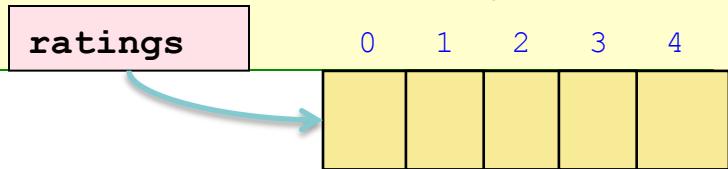
```
scores[3] = 7 + Math.random();
```

```
scores[scores.length - 1] = 9.0;
```

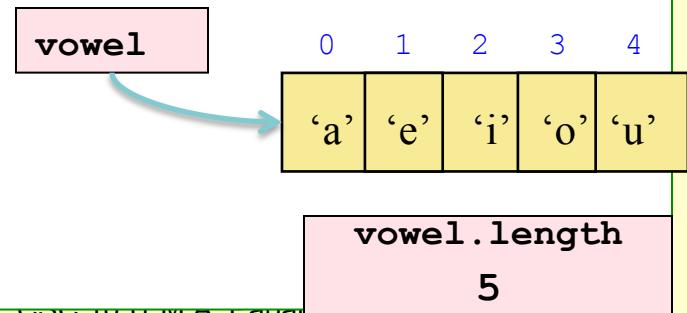
```
String answer = args[0];
```

Try this: Write some Java code to create an array

- declare and instantiate an array `ratings` that holds 5 values type `int`



- declare and instantiate an array `vowel` to hold 5 values of type `char`, then initialize its values to the vowels 'a', 'e', 'i', 'o', 'u'



scores

0	1	2	3	4	5	6	7	8	9
7.9	8.7	9.4	8.2	6.7	9.8	8.7	8.1	7.4	9.1

What gets printed?

```
System.out.println (scores[8] + 1);
```



```
System.out.println (scores[1] + scores[2]);
```



```
System.out.println (scores[1 + 2]);
```



```
System.out.println(scores[scores.length - 2]);
```



scores

0	1	2	3	4	5	6	7	8	9
7.9	8.7	9.4	8.2	6.7	9.8	8.7	8.1	7.4	9.1

Show how **scores** values change:

scores

0 1 2 3 4 5 6 7 8 9

values ? →

--	--	--	--	--	--	--	--	--	--

```
scores[4] = 1;
```

```
scores[5] = scores[0] + 1;
```

```
scores[3] = scores[1] + scores[2]);
```

```
scores[scores.length - 2]) = 5.5;
```

Processing Arrays using for-loops:

1) draw a picture of the resulting array

```
double[] mylist = new double[10];
```

```
for (int i = 0; i < 10; i++)
```

```
    mylist[i] = 0;
```

```
for (int i = 0; i < 10; i++)
```

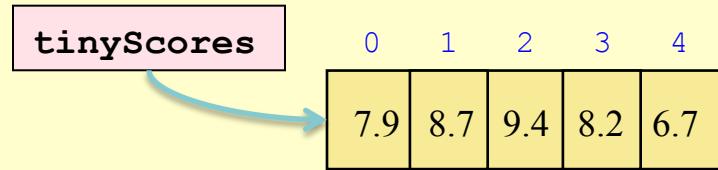
```
    mylist[i] = i;
```

Processing Arrays using for-loops:

2) Reversing through...

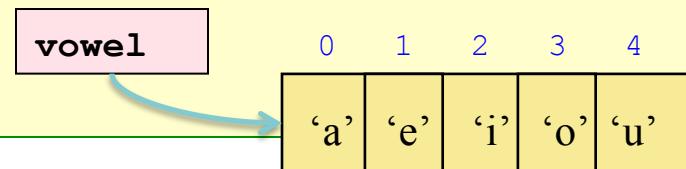
```
double[] tinyScores = new double[5];  
  
for (int i = 4; i >= 0; i--)  
  
    System.out.println(tinyScores[i]);
```

Output:



Processing Arrays using for-loops:

3) write a for-loop to print the values in the **vowel** array (going forward)



4) write a for-loop to print the values in the **vowel** array (going backward)

Bounds Checking

An array index must specify a valid element

- Example: If an array `codes` holds 100 values, it can be indexed from 0 to 99. If the value of `count` is 100, then

```
System.out.println(codes [count] ) ;
```

causes an **ArrayIndexOutOfBoundsException**

- It's common to introduce *off-by-one errors* when using arrays:

problem

```
for (int index=0; index <= 100; index++)  
    codes [index] = index*50 + epsilon;
```

Initializer Lists

- Alternative way to declare, instantiate, and initialize an array. For example:

```
int[] ratings = {4, 3, 3, 1, 4, 2, 1, 0, 3, 4};
```

```
char[] grades = {'A', 'B', 'C', 'D', 'F'};
```

- **NOTE:**
 - the `new` operator is ***not*** used
 - size of array is determined by the number of items listed
 - can only be used in the array declaration

The “for-each” Loop

- A simple way of processing every array element:

```
for (double score : scores)
    System.out.println (score);
```

NOTE:

- Only appropriate when processing all array elements starting at index 0
- It can't be used to set the array values

Another example

Try this: Use the “for each” loop to scan through an array of `int` containing ratings (range: 0 - 4) and count up how many 4's.

```
int[] ratings = {4, 3, 3, 1, 4, 3, 1, 0, 3, 4};
```

Try this: Repeat, but now count up the 0's, 1's,...
4's – Use a separate array for this

```
int[] ratings = {4, 3, 3, 1, 4, 3, 1, 0, 3, 4};
```

More array examples (see textbook):

- BasicArray.java
- Primes.java
- ReverseOrder.java
- LetterCount.java

```
*****  
// ReverseOrder.java      Author: Lewis/Loftus  
//  
// Demonstrates array index processing.  
*****  
  
import java.util.Scanner;  
  
public class ReverseOrder  
{  
    //-----  
    // Reads a list of numbers from the user, storing them in an  
    // array, then prints them in the opposite order.  
    //-----  
    public static void main (String[] args)  
    {  
        Scanner scan = new Scanner (System.in);  
  
        double[] numbers = new double[10];  
  
        System.out.println ("The size of the array: " + numbers.length);
```

continue

continue

```
for (int index = 0; index < numbers.length; index++)
{
    System.out.print ("Enter number " + (index+1) + ": ");
    numbers[index] = scan.nextDouble();
}

System.out.println ("The numbers in reverse order:");

for (int index = numbers.length-1; index >= 0; index--)
    System.out.print (numbers[index] + " ");
}
```

Sample Run

```
The size of the array: 10
```

```
Enter number 1: 18.36
```

```
Enter number 2: 48.9
```

```
Enter number 3: 53.5
```

```
Enter number 4: 29.06
```

```
Enter number 5: 72.404
```

```
Enter number 6: 34.8
```

```
Enter number 7: 63.41
```

```
Enter number 8: 45.55
```

```
Enter number 9: 69.0
```

```
Enter number 10: 99.18
```

```
The numbers in reverse order:
```

```
99.18  69.0  45.55  63.41  34.8  72.404  29.06  53.5  48.9  18.36
```

```
,
```

places numbers in an array, then prints them out **backward**

... alternatively, we could place the numbers in the array **backward** and then print them **forward**

Another example: Computing letter frequency counts

Sample run:

Enter a sentence:

In Casablanca, Humphrey Bogart never says "Play it again,
Sam."

A: 0	a: 10
B: 1	b: 1
C: 1	c: 1
D: 0	d: 0
E: 0	e: 3

...



Let's write a
program to
do this

```
*****  
// LetterCount.java          Author: Lewis/Loftus  
//  
// Demonstrates the relationship between arrays and strings.  
*****  
  
import java.util.Scanner;  
  
public class LetterCount  
{  
    //-----  
    // Reads a sentence from the user and counts the number of  
    // uppercase and lowercase letters contained in it.  
    //-----  
    public static void main (String[] args)  
    {  
        final int NUMCHARS = 26;  
  
        Scanner scan = new Scanner (System.in);  
  
        int[] upper = new int[NUMCHARS];  
        int[] lower = new int[NUMCHARS];  
  
        char current; // the current character being processed  
        int other = 0; // counter for non-alphabetics
```

continue

continue

```
System.out.println ("Enter a sentence:");
String line = scan.nextLine();

// Count the number of each letter occurrence
for (int ch = 0; ch < line.length(); ch++)
{
    current = line.charAt(ch);
    if (current >= 'A' && current <= 'Z')
        upper[current-'A']++;
    else
        if (current >= 'a' && current <= 'z')
            lower[current-'a']++;
        else
            other++;
}
// Print the results
System.out.println ();
for (int letter=0; letter < upper.length; letter++)
{
    System.out.print ( (char) (letter + 'A') );
    System.out.print (" : " + upper[letter]);
    System.out.print ("\t\t" + (char) (letter + 'a') );
    System.out.println (" : " + lower[letter]);
}

System.out.println ();
System.out.println ("Non-alphabetic characters: " + other);
}
```

Sample Run

Enter a sentence:

In Casablanca, Humphrey Bogart never says "Play it again, Sam."

A: 0	a: 10
B: 1	b: 1
C: 1	c: 1
D: 0	d: 0
E: 0	e: 3
F: 0	f: 0
G: 0	g: 2
H: 1	h: 1
I: 1	i: 2
J: 0	j: 0
K: 0	k: 0
L: 0	l: 2
M: 0	m: 2
N: 0	n: 4
O: 0	o: 1
P: 1	p: 1
Q: 0	q: 0

continue

Sample Run (continued)

R: 0	r: 3
S: 1	s: 3
T: 0	t: 2
U: 0	u: 1
V: 0	v: 1
W: 0	w: 0
X: 0	x: 0
Y: 0	y: 3
Z: 0	z: 0

Non-alphabetic characters: 14

What does it mean to “copy an array”?

- Suppose we have two arrays:

```
int[] a = {147, 323, 89, 933};
```

```
int[] b = {100, 200, 300, 400};
```

Copying elements vs. copying array variables:

```
for (int i=0; i<a.length; i++)
    a[i] = b[i];
```

```
a = b;
```

Afterwards, what is the effect of the following?

```
a[1] = 1111;
b[2] = 2222;
```

1) Copying elements:

a

	0	1	2	3
	147	323	89	933

b

	0	1	2	3
	100	200	300	400

What changes?

```
for (int i=0; i<a.length; i++)
    a[i] = b[i];
```

```
a[1] = 1111;
b[2] = 2222;
```

2) Copying array variables:

a

	0	1	2	3
	147	323	89	933

b

	0	1	2	3
	100	200	300	400

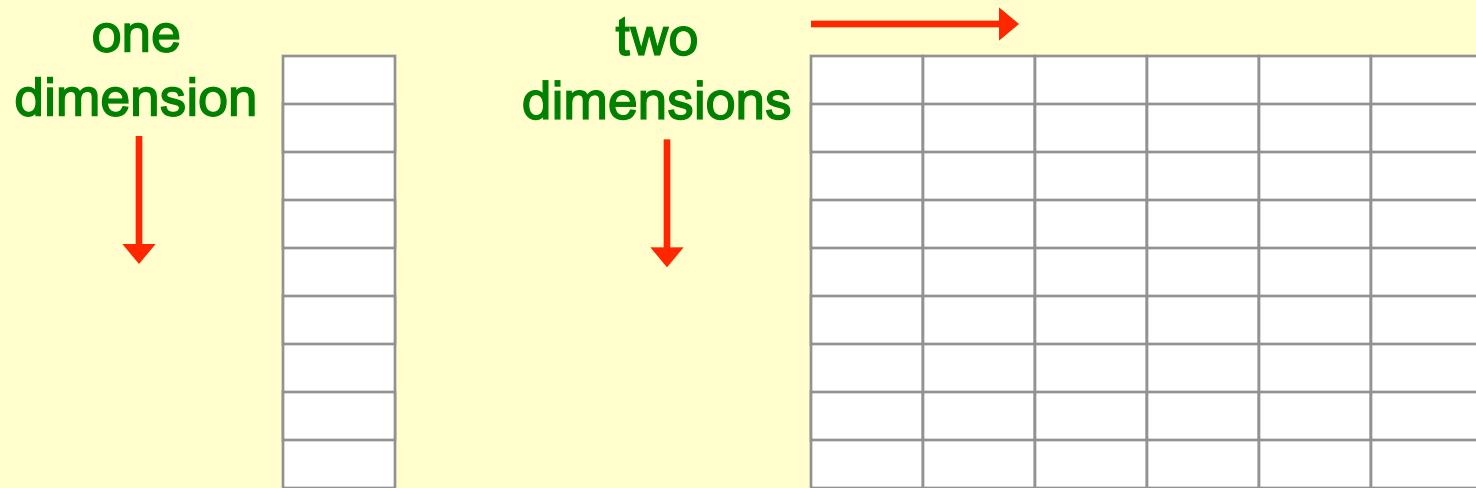
What changes?

```
a = b;
```

```
a[1] = 1111;  
b[2] = 2222;
```

Two-Dimensional Arrays

- A *one-dimensional array* stores a list of elements
- A *two-dimensional array* can be thought of as a table of elements, with rows and columns



Two-Dimensional Arrays

- To be precise, in Java a two-dimensional array is an array of arrays
- A two-dimensional array is declared by specifying the size of each dimension separately:

```
int[][] table = new int[12][50];
```

- An array element is referenced using two index values:

```
value = table[3][6]
```
- The array stored in one row can be specified using one index

declaration

2D Arrays - Overview

```
double[][] courseGrade = new double[3][10];
```

2D array element

courseGrade [1] [4]

courseGrade

0
1
2

0	1	2	3	4	5	6	7	8	9
7.9	8.7	9.4	8.2	6.7	9.8	8.7	8.1	7.4	9.1

0	1	2	3	4	5	6	7	8	9
9.3	5.8	6.9	5.5	9.0	8.3	7.7	9.2	9.8	8.2

0	1	2	3	4	5	6	7	8	9
8.9	8.0	8.4	6.2	7.7	7.3	9.6	6.1	7.8	7.3

array element (a row)

courseGrade [2]

Two-Dimensional Arrays

Expression	Type	Description
<code>table</code>	<code>int[][]</code>	2D array of integers, or array of integer arrays
<code>table[5]</code>	<code>int[]</code>	array of integers
<code>table[5][12]</code>	<code>int</code>	integer

- See [TwoDArray.java](#)
- See [SodaSurvey.java](#)

```
//*****  
//  TwoDArray.java          Author: Lewis/Loftus  
//  
//  Demonstrates the use of a two-dimensional array.  
//*****  
  
public class TwoDArray  
{  
    //-----  
    //  Creates a 2D array of integers, fills it with increasing  
    //  integer values, then prints them out.  
    //-----  
    public static void main (String[] args)  
    {  
        int[][] table = new int[5][10];  
  
        // Load the table with values  
        for (int row=0; row < table.length; row++)  
            for (int col=0; col < table[row].length; col++)  
                table[row][col] = row * 10 + col;  
  
        // Print the table  
        for (int row=0; row < table.length; row++)  
        {  
            for (int col=0; col < table[row].length; col++)  
                System.out.print (table[row][col] + "\t");  
            System.out.println();  
        }  
    }  
}
```

```
*****  
// TwoDArray.java          Author: Lewis/Loftus
```

Output

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49

```
-----  
public static void main (String[] args)  
{  
    int[][] table = new int[5][10];  
  
    // Load the table with values  
    for (int row=0; row < table.length; row++)  
        for (int col=0; col < table[row].length; col++)  
            table[row][col] = row * 10 + col;  
  
    // Print the table  
    for (int row=0; row < table.length; row++)  
    {  
        for (int col=0; col < table[row].length; col++)  
            System.out.print (table[row][col] + "\t");  
        System.out.println();  
    }  
}
```

Multidimensional Arrays

- An array can have many dimensions – if it has more than one dimension, it is called a *multidimensional array*
- Each dimension subdivides the previous one into the specified number of elements
- Each dimension has its own length constant
- Because each dimension is an array of array references, the arrays within one dimension can be of different lengths
 - these are sometimes called *ragged arrays*