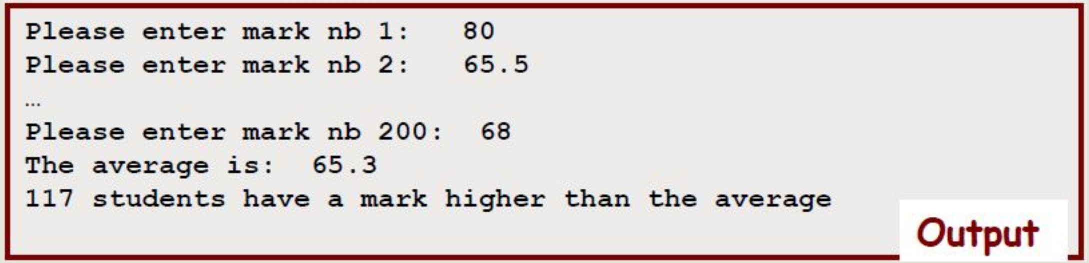
Arrays

Programming in Java

Contents

1- Arrays

- Problem:
  - read 200 marks and compute how many are < the average …

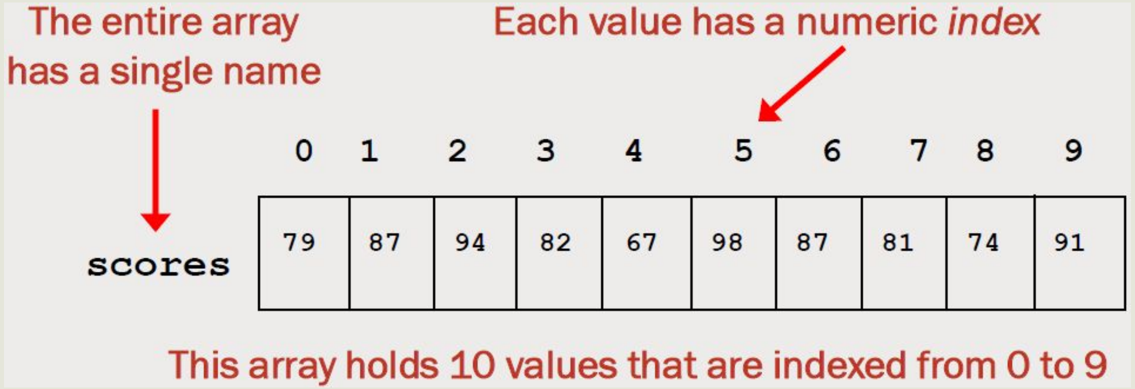


```
Please enter mark nb 1:   80
Please enter mark nb 2:   65.5
…
Please enter mark nb 200:  68
The average is:  65.3
117 students have a mark higher than the average
```

  - Need to store 200 variables !!!
- Solution:
  - use an array!
  - an array helps us organize large amounts of information

1- Arrays

- An *array* is an ordered list of elements of the same type:



The entire array has a single name
Each value has a numeric *index*
0 1 2 3 4 5 6 7 8 9
scores
79 87 94 82 67 98 87 81 74 91
This array holds 10 values that are indexed from 0 to 9

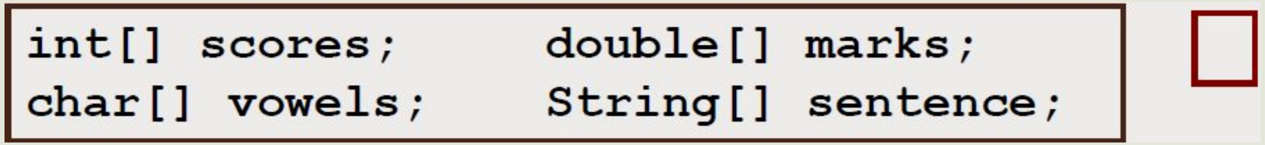
- An array of size n is indexed from 0 to n - 1

1- Arrays

- The elements of an array can be:
  - a primitive type or
  - an object reference (we'll see this later)

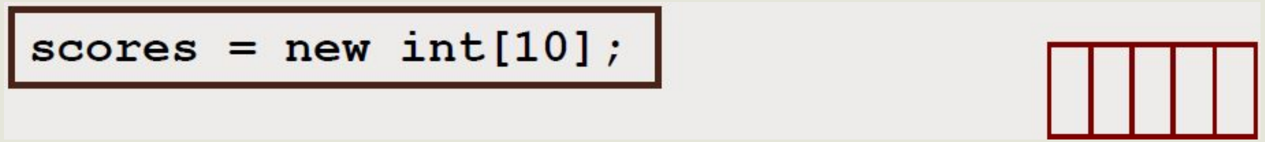
1- Arrays: Declaring and creating arrays

- Declaring the reference:
  - syntax: *type_of_elements[] name_of_array;*



```
int[] scores;        double[] marks;
char[] vowels;       String[] sentence;
```

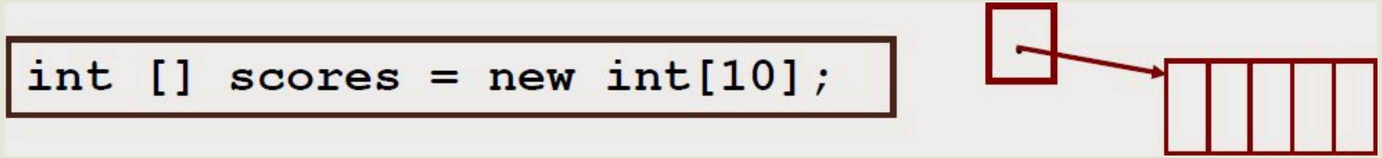
- Creating the elements:
  - syntax: *name_of_array = new type_of_elements[size];*



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

1- Arrays: Declaring and creating arrays

- Declaration + creation:



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

1- Arrays: Declaring and creating arrays

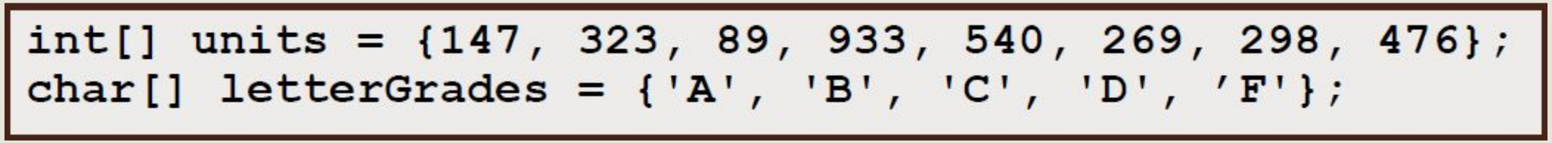
- Size of the array:
  - must be an integer expression (constant or variable)

```
double[] price;
int nbItems = Keyboard.readInt();
price = new double[nbItems];
```

- Array initialization:
  - every element is initialized to zero
    - int , double ----> 0
    - boolean ----> false
    - references ----> null

1- Arrays: Initializer lists

- We can declare and initialize an array in one step:



```
int[] units = {147, 323, 89, 933, 540, 269, 298, 476};
char[] letterGrades = {'A', 'B', 'C', 'D', ’F'};
```

- Note:
    - no size value is specified (size = nb of elements specified)
    - the **new** operator is not used

1- Arrays: Just checking ...

Which of the following initializer lists correctly initializes the indexed variables of an array named myDoubles?

A. **double myDoubles[double] = {0.0, 1.0, 1.5, 2.0, 2.5};**

B. **double myDoubles[5] = new double(0.0, 1.0, 1.5, 2.0, 2.5);**

C. **double[] myDoubles = {0.0, 1.0, 1.5, 2.0, 2.5};**

D. **array myDoubles[double] = {0.0, 1.0, 1.5, 2.0, 2.5};**

E. **All of the above are valid**

1- Arrays: Just checking ...

Given the declaration

```
int[] alpha = new int[75];
```

the valid range of index values for alpha is:

A. 0 through 75

B. 0 through 74

C. 1 through 75

D. 1 through 74

E. 1 through 76

1- Arrays: Access to an element

- Syntax: *nameOfArray[indexOfElement]*

```
double[] scores = new double[10];
scores[2] = 55.5;
scores[0] = scores[2] + 2;
double mean = (scores[0] + scores[2])/2;
System.out.print(mean);
```

```
???
```

Output

- assignment to elements… one by one

```
fill an array with:
10 20 30 40 50
then display array
```

To do

```
final int LIMIT = 5;
int[] list = new int[LIMIT];

???
```

1- Arrays: Automatic bounds checking

- The index must be [0 … length 1]
- The Java interpreter checks for you…
- if an array index is out of bounds ---->

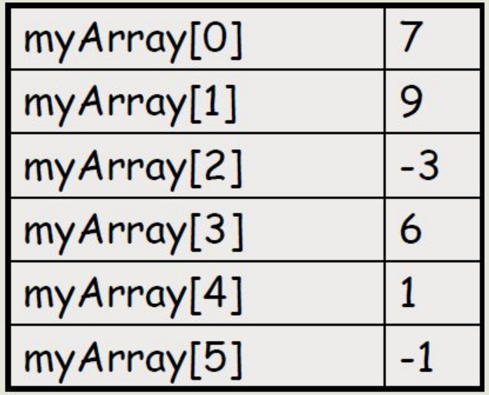
  ArrayIndexOutOfBoundsException

```
double [] codes = new double[100];
int count = 100;
System.out.println(codes[count]);
```

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

1- Arrays: Just checking ...

Consider the following array:



myArray[0]	7
myArray[1]	9
myArray[2]	-3
myArray[3]	6
myArray[4]	1
myArray[5]	-1

What is the value of

myArray[myArray[1] - myArray[0]]

A. 2
B. 9
C. -3
D. 6
E. 7

1- Arrays: Just checking ...

Given the declarations

```
int[] status = new int[10];
int i;
```

which of the following loops correctly fills the array status with 1s?

A.
```
for (i = 0; i <= 10; i++)
    status[i] = 1;
```

B.
```
for (i = 0; i < 10; i++)
    status[i] = 1;
```

C.
```
for (i = 1; i <= 10; i++)
    status[i] = 1;
```

D.
```
for (i = 1; i < 10; i++)
    status[i] = 1;
```

E.
```
for (i = 1; i <= 11; i++)
    status[i] = 1;
```

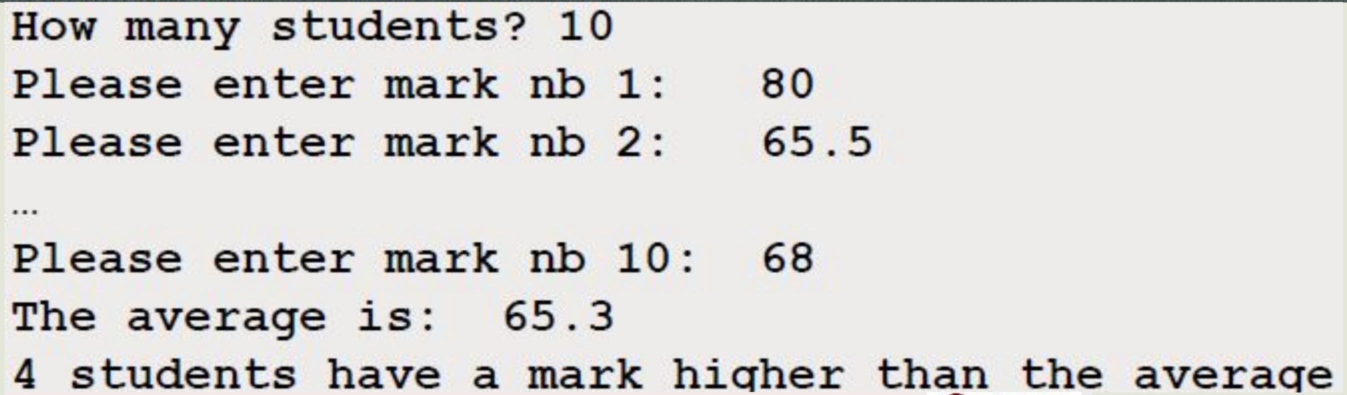
1- Arrays: The **length** instance variable

- An array is an object
- **length** is a public constant (an attribute) that gives the number of elements in the array

```
int[] score = new int [10];
System.out.print(scores.length);
```

```
double[] numbers = new double[10];
System.out.println ("The size of the array: "+
   numbers.length);

// to fill the array with user input



// to display in reverse order
```

1- Arrays: Example



```
How many students? 10
Please enter mark nb 1:   80
Please enter mark nb 2:   65.5
...
Please enter mark nb 10:  68
The average is:  65.3
4 students have a mark higher than the average
```

Output

Variables needed?

Algorithm?

1- Arrays: Example

```
Enter the age of the 30 children (from 0 to 10):
2  3  0  1 … 9
3 children are 0 years old
2 children are 1 years old
…
4 children are 10 years old
```

Variables needed?

Algorithm?

2- Sorting

- Sorting is the process of arranging a list of items in a particular order
- There are many algorithms for sorting a list of items
- some are more efficient, some are more intuitive
  - Selection Sort
  - Insertion Sort
  - Bubble Sort
  - Quicksort
  - ...

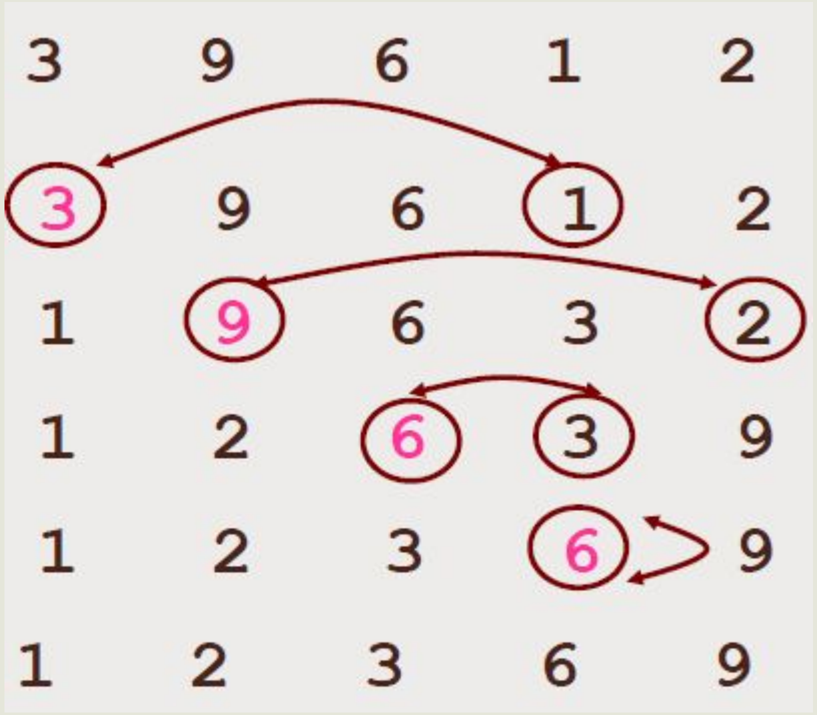
2- Sorting: Selection Sort

- The approach:
  - select a value and put it in its final place into the list
  - repeat for all other values

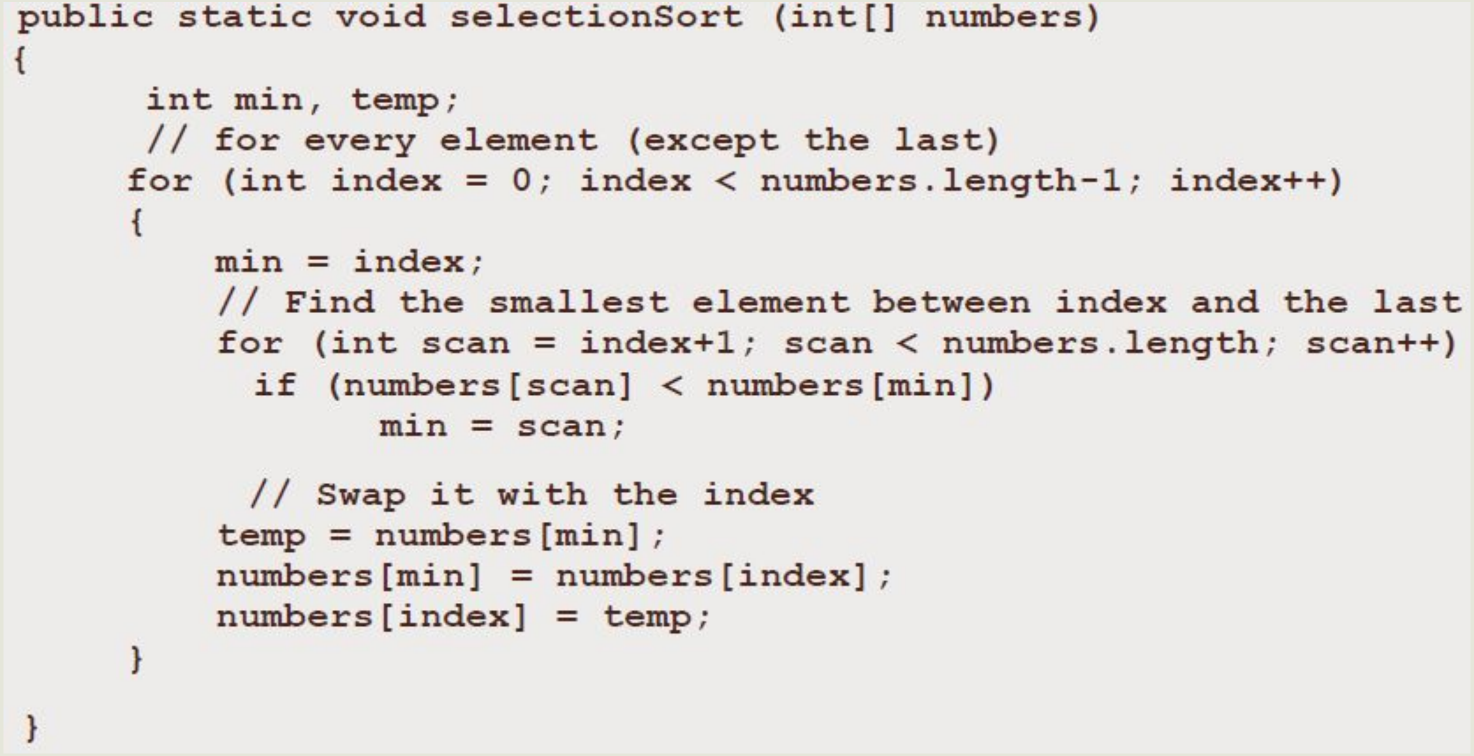
- In more detail:
  - find the smallest value in the array
  - switch it with the value in the first position
  - find the next smallest value in the array
  - switch it with the value in the second position
  - repeat until all values are in their proper places

2- Sorting: Selection Sort: Example

- Original:
- Smallest is 1:
- Smallest is 2:
- Smallest is 3:
- Smallest is 6:
- Sorted array:



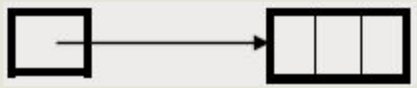
2- Sorting: Selection Sort: Example



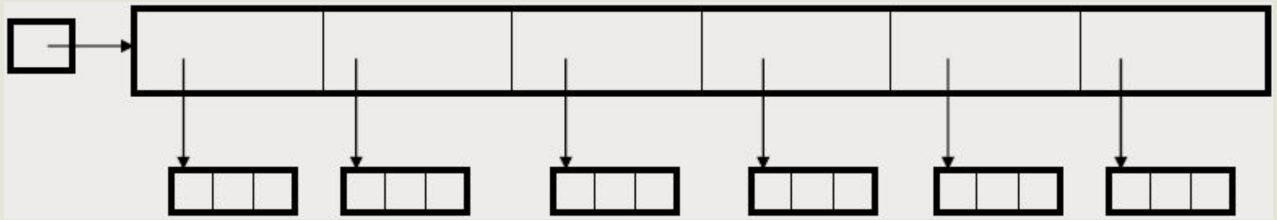
```
public static void selectionSort (int[] numbers)
{
        int min, temp;
        // for every element (except the last)
       for (int index = 0; index < numbers.length-1; index++)
       {
           min = index;
           // Find the smallest element between index and the last
           for (int scan = index+1; scan < numbers.length; scan++)
             if (numbers[scan] < numbers[min])
                    min = scan;

             // Swap it with the index
           temp = numbers[min];
           numbers[min] = numbers[index];
           numbers[index] = temp;
       }

}
```

3- Multidimensional arrays

- A one-dimensional array
  - stores a list of elements of simple type (primitive or reference)

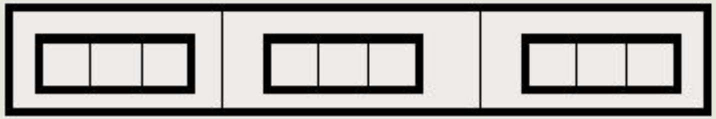


- A two-dimensional array
  - stores a list of elements, where each element is a 1-D array of simple type



3- Multidimensional arrays

- a 2-D array is really a 1-D array of references to 1-D arrays
- so the arrays within one dimension can be of different lengths (called ragged arrays)
- not :



3- Multidimensional arrays: two-dimensional

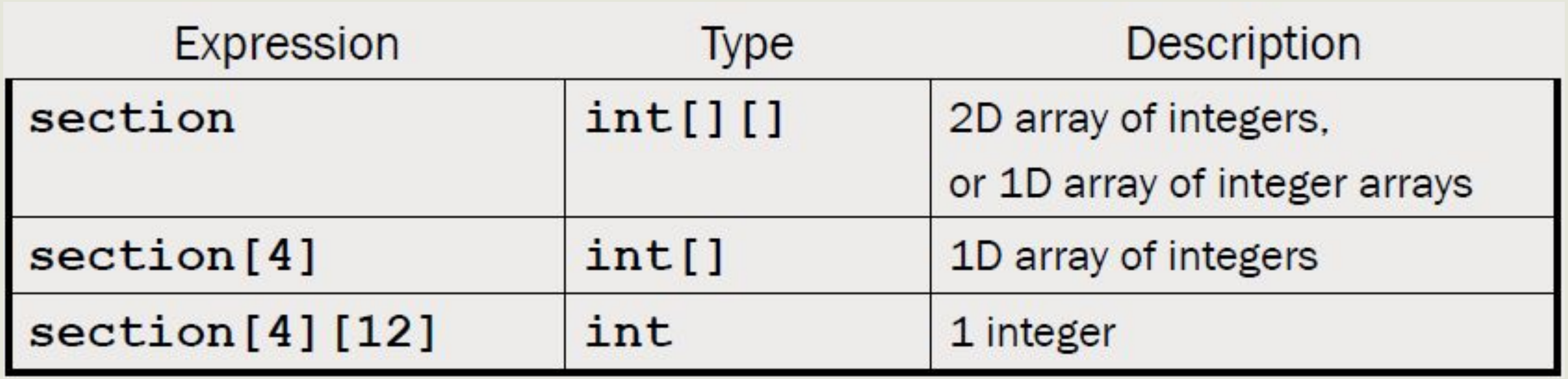
- Declaration:

```
double[] student = new double[5]; // 1-D 5 tests for 1 student
double[][] section = new double[5][80]; // 2-D 80 students per section
double[][][] course = new double[3][5][80]; // 3-D 5 sections per course
```

- Access to an element

```
value = section[3][5]
```

3- Multidimensional arrays: two-dimensional



Expression	Type	Description
`section`	`int[][]`	2D array of integers, or 1D array of integer arrays
`section[4]`	`int[]`	1D array of integers
`section[4][12]`	`int`	1 integer

3- Multidimensional arrays: Just checking ...

Given the declaration

```
double costOfGoods[ ][ ][ ]= new double [8][2][7];
```

how many float components does **costOfGoods** have?

A. 8

B. 70

C. 72

D. 112

E. 17

3- Multidimensional arrays: Just checking ...

Given the declarations

```
double x[] = new double[300];
double y[][] = new double[75][4];
double z[] = new double[79];
```

which of the following statements is true?

A. x has more components than y.

B. y has more components than x.

C. y and z have the same number of components.

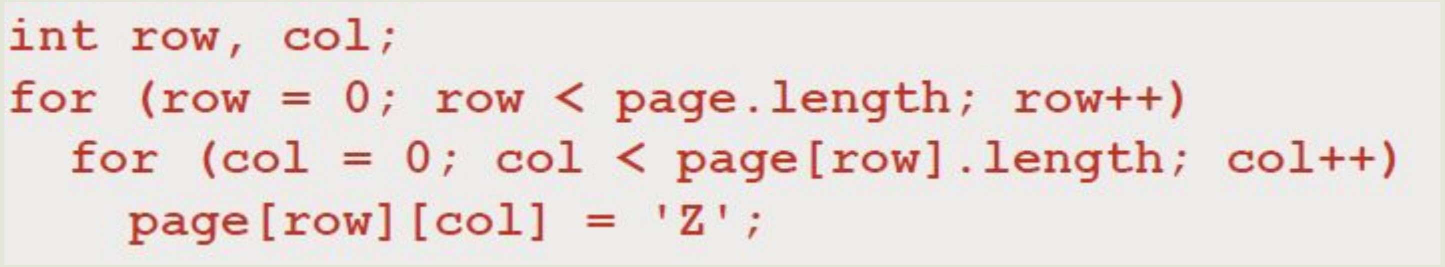
D. x and y have the same number of components.

E. a and c above

3- Multidimensional arrays: **length**

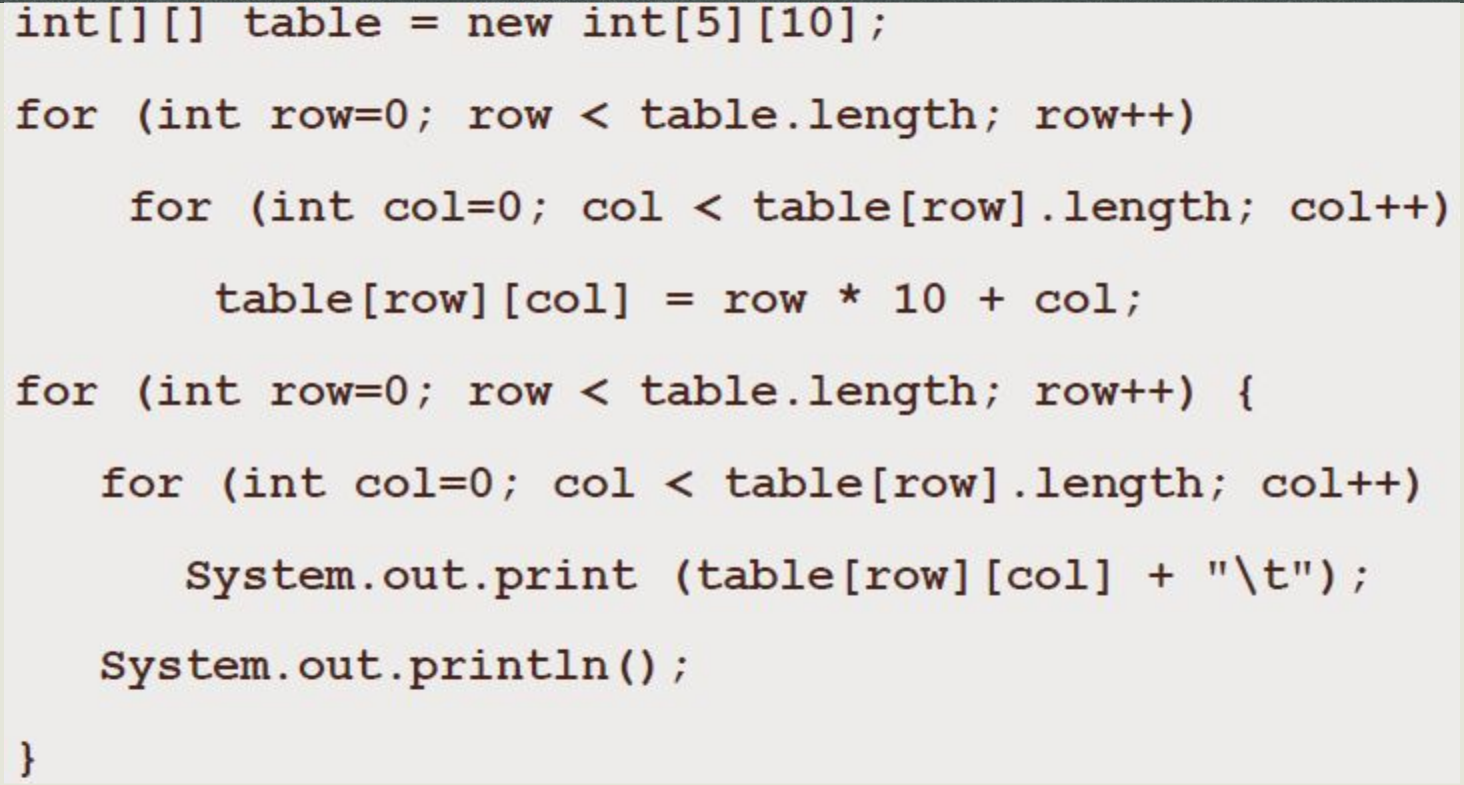
char[][] page = new char[30][100];

- **length** does not give the total number of indexed variables
  - **page.length** is equal to 30
  - **page[0].length** is equal to 100



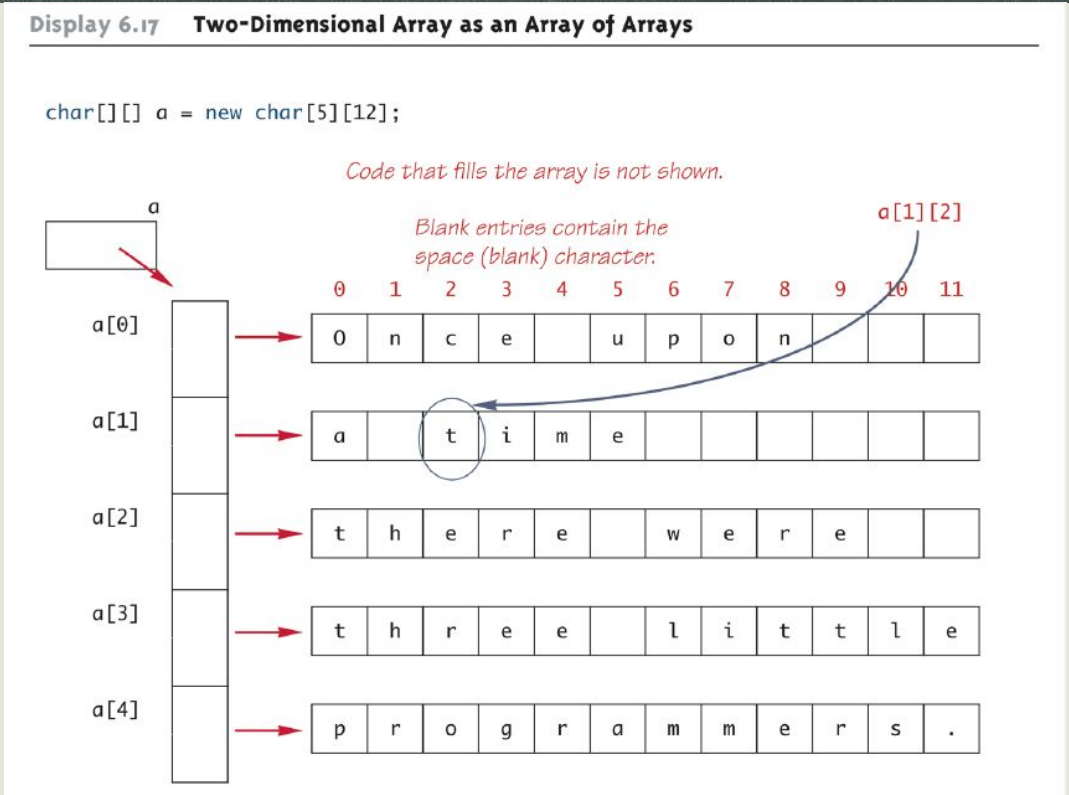
```
int row, col;
for (row = 0; row < page.length; row++)
  for (col = 0; col < page[row].length; col++)
    page[row][col] = 'Z';
```

3- Multidimensional arrays: Example



```
int[][] table = new int[5][10];
for (int row=0; row < table.length; row++)
    for (int col=0; col < table[row].length; col++)
       table[row][col] = row * 10 + col;
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();
}
```

3- Multidimensional arrays: two-dimensional


Display 6.17 **Two-Dimensional Array as an Array of Arrays**

```
char[][] a = new char[5][12];
```

Code that fills the array is not shown.
Blank entries contain the space (blank) character.
a[1][2]
a
a[0]
a[1]
a[2]
a[3]
a[4]

	0	1	2	3	4	5	6	7	8	9	10	11
a[0]	O	n	c	e		u	p	o	n			
a[1]	a		t	i	m	e						
a[2]	t	h	e	r	e		w	e	r	e		
a[3]	t	h	r	e	e		l	i	t	t	l	e
a[4]	p	r	o	g	r	a	m	m	e	r	s	.

3- Multidimensional arrays: Ragged arrays

```
double[][] a = new double[3][5];
```

- is equivalent to:

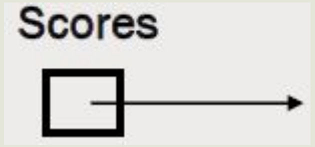
```
double [][] a;
a = new double[3][];
a[0] = new double[5];
a[1] = new double[5];
a[2] = new double[5];
```

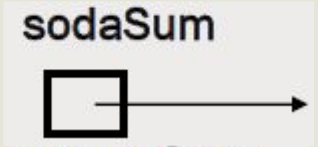
- So we could have:

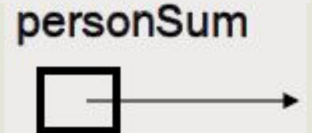
```
double [][] a = new double[3][];
a[0] = new double[5];
a[1] = new double[10];
a[2] = new double[4];
```

3- Multidimensional arrays: Ragged arrays: Example

```
int[][] scores = { {1, 2, 3}, {1, 1, 1}, {3, 4, 5}, {10, 20, 30}};

final int SODAS = scores.length;
final int PEOPLE = scores[0].length;

int[] sodaSum = new int[SODAS];
int[] personSum = new int[PEOPLE];
............
```

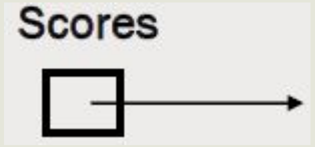


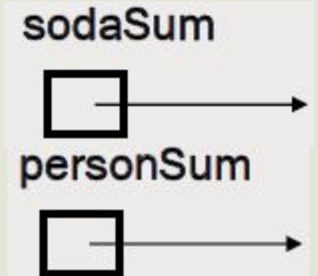




3- Multidimensional arrays: Ragged arrays: Example - cont'd

```
……
for (int soda=0; soda < SODAS; soda++)
   for (int person=0; person < PEOPLE; person++) {
      sodaSum[soda] += scores[soda][person];
      personSum[person] += scores[soda][person];

   }
……
```





3- Multidimensional arrays: Ragged arrays: Example - cont’d

```
……

for (int soda=0; soda < SODAS; soda++)

   System.out.println("Soda #" + (soda+1) + ": " +
   sodaSum[soda]/PEOPLE);


for (int person =0; person < PEOPLE; person++)

   System.out.println ("Person #"+ (person+1) +": "+
   personSum[person]/SODAS);
```

Output