

# ARRAYS

---

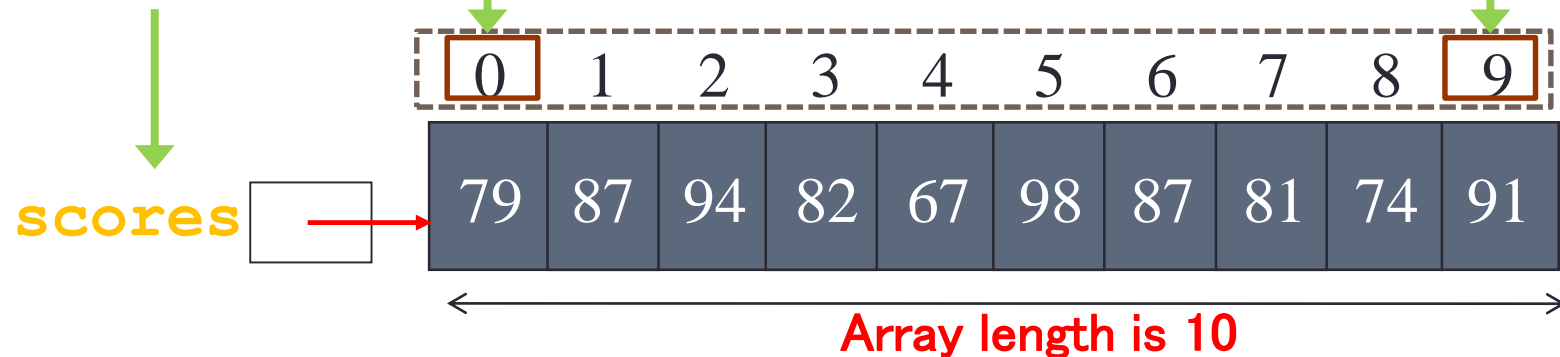
# Arrays

- An array is a container object that holds a fixed number of values of a single type.
- The length of an array is established when the array is created. After creation, its length is fixed.
- Each item in an array is called an element, and each element is accessed by its numerical index.
- As shown in the below illustration, numbering index begins with 0 and ends with its length-1.

# Arrays

- An *array* is an ordered list of values

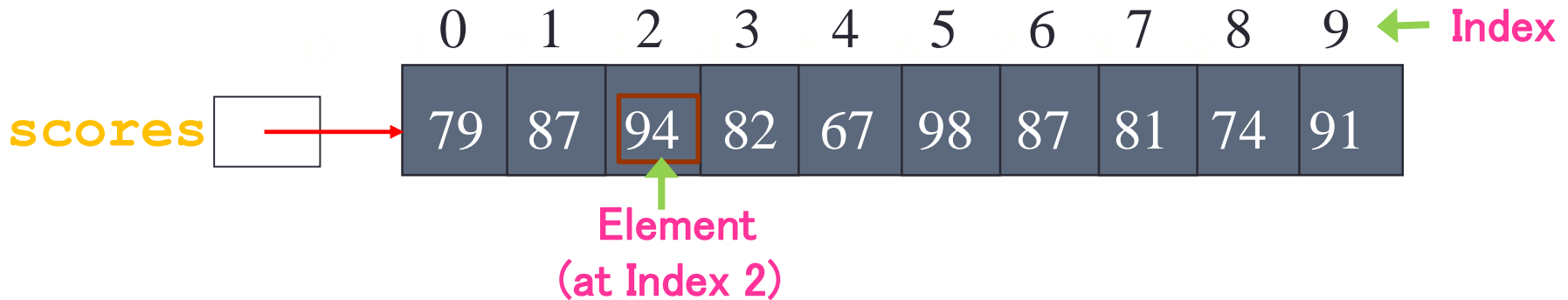
The entire array  
has a single name



An array of size  $N$  is indexed from zero to  $N-1$

This array holds 10 values that are indexed from 0 to 9

# Arrays



- A particular value in an array is referenced using the array name followed by the index in brackets, For example :

`scores[2]`

refers to the value 94 (the 3rd value in the array)

# Arrays

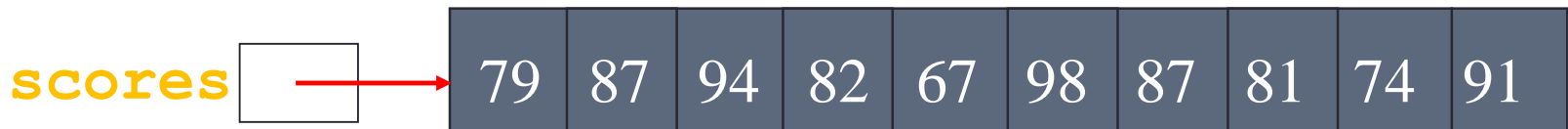
- For example, an array element can be assigned a value, printed, or used in a calculation :

```
scores[2] = 89;
```

```
scores[first] = scores[first] + 2;
```

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

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



# Declaring and Creating Separately

- Declaring Array

- `datatype[] arrayVar;`

Example: `double[] myList;`

`int[] scores, x, y; //scores, x, and y are array`

- `datatype arrayVar[];`

Example: `double myList[];`

`int scores[], x, y[]; //only scores and y are array`

- Creating Array

`arrayVar = new datatype[arraySize];`

Example:

`myList = new double[20];`

`scores = new int[10];`

`y = new int[100];`

# Declaring and Creating in One Step

- `datatype[] arrayVar = new datatype[arraySize];`

## Example:

```
double[] myList = new double[MAX];  
float[] values = new float[35];  
boolean status[] = new boolean[21];
```

# Default Values

- When an array is created, its elements are assigned the default value of
  - 0 for the integer data types
  - 0.0 for the floating point data types
  - `'\u0000'` for char types, and
  - `false` for boolean types.



# Bounds Checking

- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element
- That is, the index value must be in range 0 to N-1
- The Java interpreter throws an *ArrayIndexOutOfBoundsException* if an array index is out of bounds
- This is called automatic bounds checking

# Bounds Checking

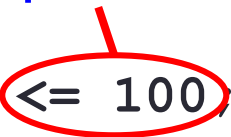
- For example, if the array `codes` can hold 100 values, it can be indexed using only the numbers 0 to 99
- If the value of `count` is 100, then the following reference will cause an exception to be thrown:

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

- 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;
```



# Bounds Checking

- Each array object has a public constant called `length` that stores the size of the array
- It is referenced using the array name,

`arrayVar.length`

- Note that `length` holds the number of elements, not the largest index

# Array Example

```
public class Test {  
    public static void main(String[] args) {  
        int[] values = new int[5];  
        for (int i = 1; i < values.length; i++) {  
            values[i] = i + values[i-1];  
        }  
        values[0] = values[1] + values[4];  
    }  
}
```

After the array is created

0	11
1	1
2	3
3	6
4	10

# Initializer Lists

- An initializer list can be used to instantiate and fill an array in one step
- The values are delimited by braces and separated by commas
- Examples:

```
int[] units = {147, 323, 89, 933, 540,  
               269, 97, 114, 298, 476};
```

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

# Initializer Lists

- Note that when an initializer list is used:
  - the new operator is not used
  - no size value is specified
- The size of the array is determined by the number of items in the initializer list
- An initializer list can be used only in the array declaration

# Arrays of Objects

- The elements of an array can be object references
- The following declaration reserves space to store 5 references to `String` objects

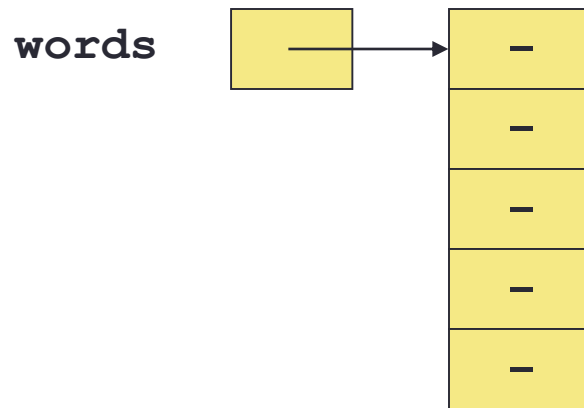
```
String[] words = new String[5];
```

- It does NOT create the `String` objects themselves
- Initially an array of objects holds `null` references
- Each object stored in an array must be instantiated separately such as

```
words[0]=new String("friendship"); or  
words[0]="friendship";
```

# Arrays of Objects

- The `words` array when initially declared:



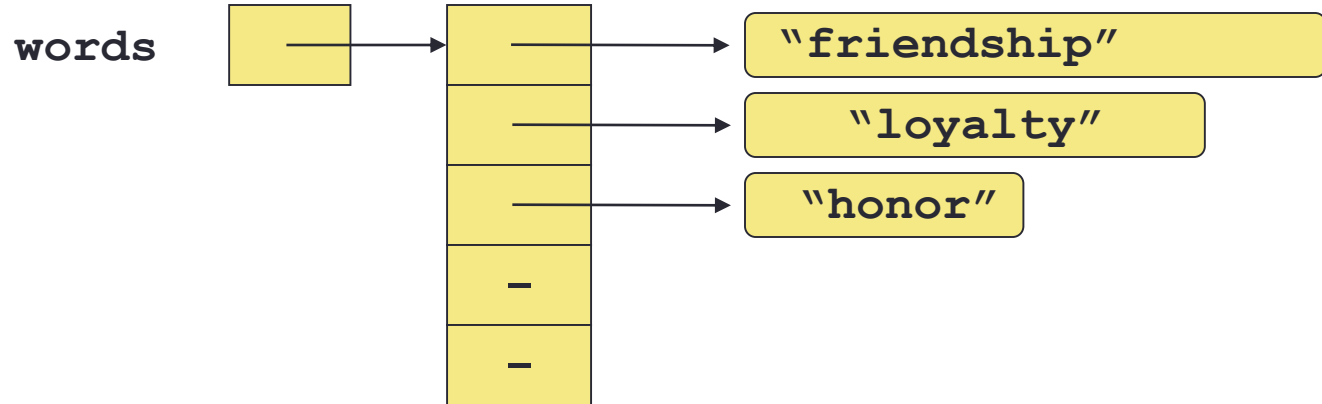
At this point, the following reference would throw a `NullPointerException`:

```
System.out.println (words[0].length());
```



# Arrays of Objects

- After some `String` objects are created and stored in the array:



```
String[] words = new String[5];  
words[0] = "friendship";  
words[1] = new String("loyalty");  
words[2] = "honor";
```

# Initializer List of Object Arrays

- Keep in mind that `String` objects can be created using literals
- The following declaration creates an array object called `verbs` and fills it with four `String` objects

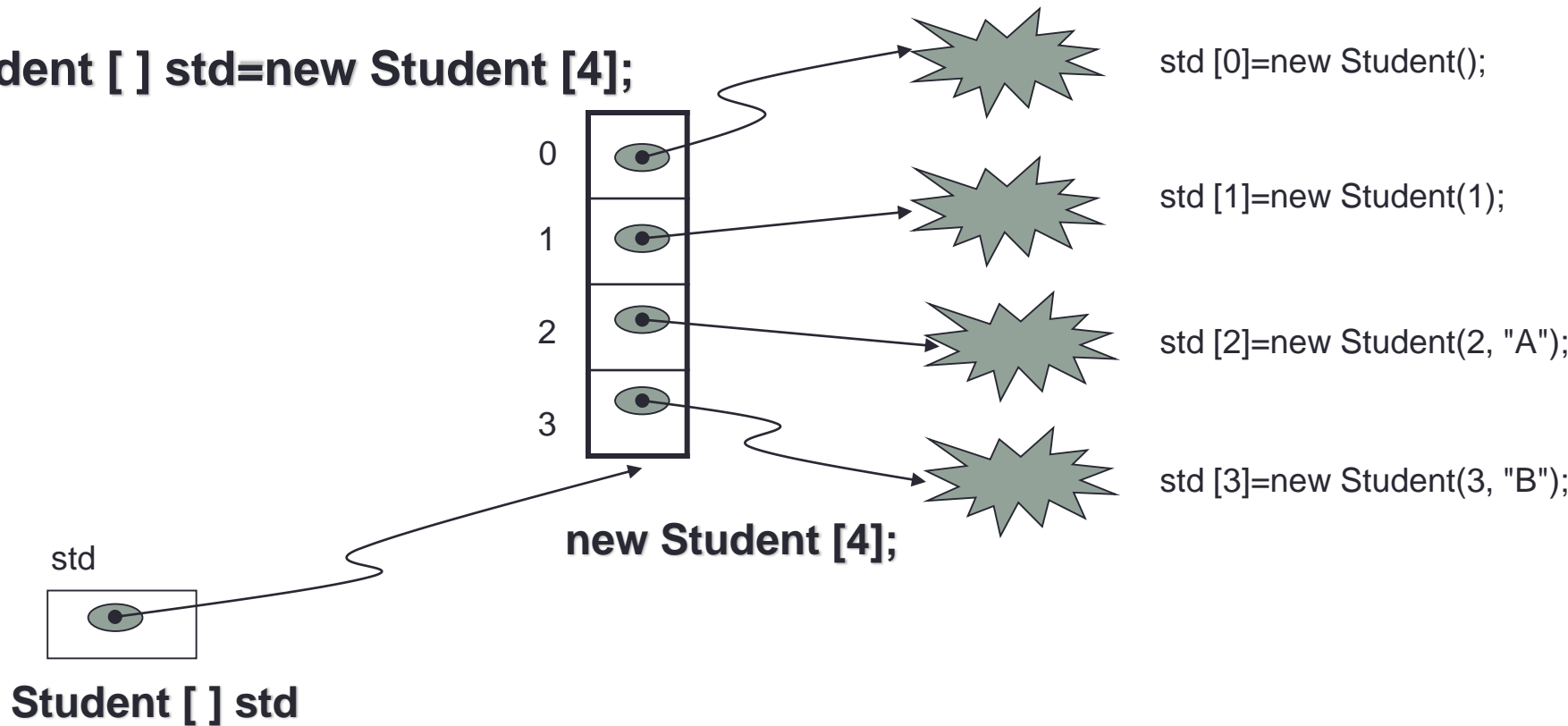
```
String[] verbs = {"play", "work", "eat", "sleep"};
```

**OR**

```
String[] verbs = {new String("play"), new String("work"),  
new String("eat"), new String("sleep")};
```

# Initializer List of Object Arrays

**Student [ ] std=new Student [4];**



# Two Methods for Creating Object Arrays

**1. Student [ ] std=new Student [4];**

std [0]=new Student();

std [1]=new Student(1);

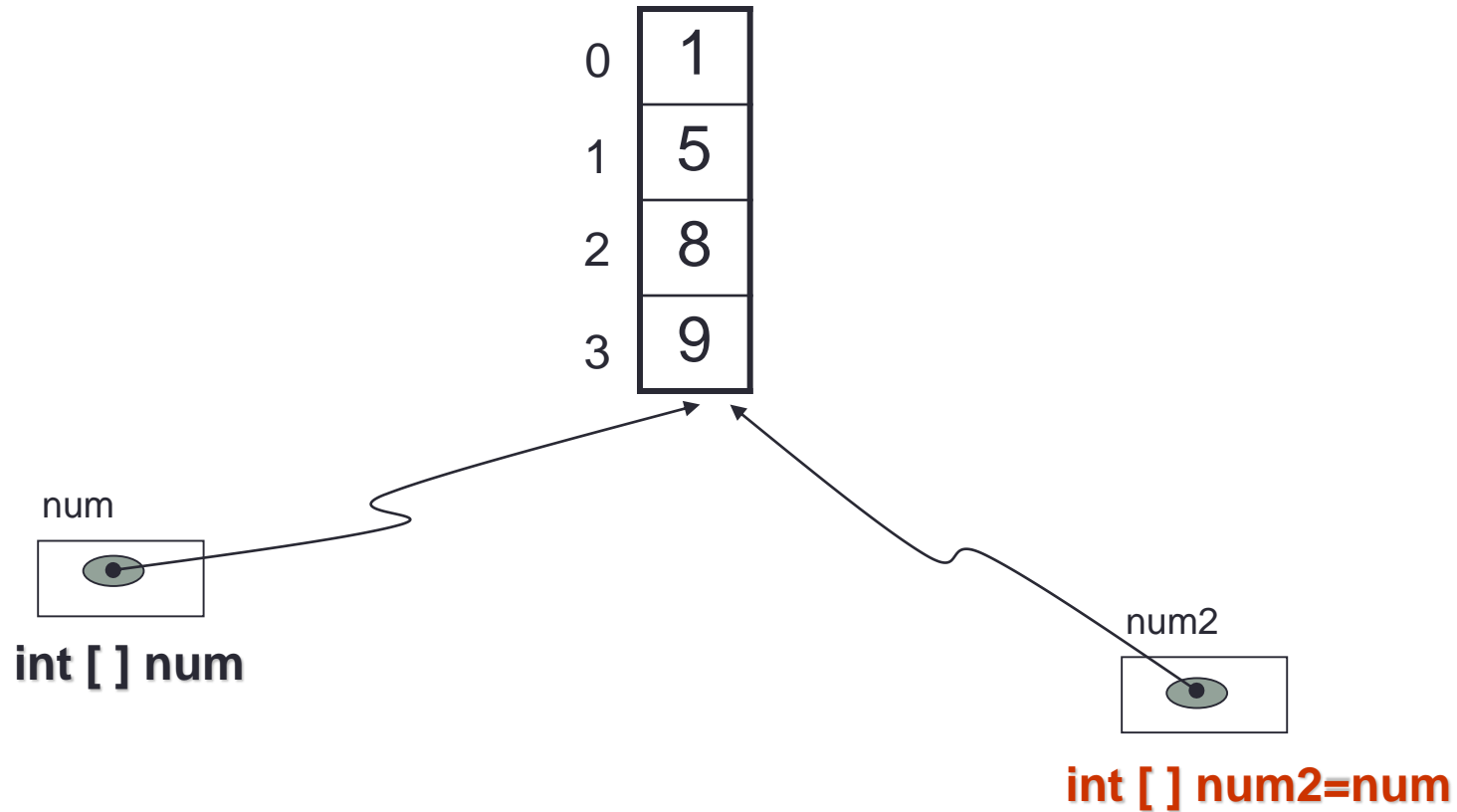
std [2]=new Student(2, "A");

std [3]=new Student(3, "B");

**2. Student std[ ] ={new Student(), new Student(1),new Student(2, "A"), new Student(3, "B")};**

# Array Assignment

```
int [ ] num =new int[4];
```




# Command-Line Arguments

- The signature of the main method indicates that it takes an array of String objects as a parameter
- These values come from command-line arguments that are provided when the interpreter is invoked
- For example, the following invocation of the interpreter passes three String objects into main:

C:\> java StateEval 

pennsylvania	texas	arizona
--------------	-------	---------



index 0      index 1      index 2

- These strings are stored at indexes 0-2 of the array parameter of the main method

# Command-Line Arguments: Example

```
public class Adder {  
    public static void main(String[] args) {  
        if (args.length == 2) {  
            int x = Integer.parseInt(args[0]) ;  
            int y = Integer.parseInt(args[1]) ;  
            System.out.println("Result = " + (x+y));  
        }  
        else  
            System.out.println("Using: java Adder value1 value2");  
    }  
}
```

# For each

- The another version of the for loop can be used when processing array elements

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

**This is only appropriate when processing all array elements from top (lowest index) to bottom (highest index)**



# For each Example

```
int[] scores = {2, 5, 9, 11, 25};  
for (int score : scores)  
    System.out.println (score);
```

```
Student stdArr[] ={new Student(), new Student(1), new  
    Student(2, "A"), new Student(3, "B")};  
for(Student s: stdArr)  
    System.out.println(s);
```

```
double[] values = {2, 5, 9, 11, 25};  
double sum = 0;  
for (double v : values)  
    sum = sum + v;  
System.out.println("Average: "+ (sum/values.length));
```

# Arrays as Parameters

- An entire array can be passed as a parameter to a method
- Like any other object, the reference to the array is passed, making the formal and actual parameters aliases of each other
- Therefore, changing an array element within the method changes the original
- An individual array element can be passed to a method as well, in which case the type of the formal parameter is the same as the element type

# Example: ArrayAsParameter.java

```
public class ArrayAsParameter {
    public static void printArray(int a[]){
        /*for(int i=0;i<a.length;i++)
            System.out.println("a["+i+"] = "+a[i]);*/
        for(int i: a){
            System.out.println(i);
        }
    }
    public static int[] sumArray(int a[], int b[]){
        int[] result=new int[a.length];
        for(int i=0;i<a.length;i++)
            result[i]=a[i]+b[i];
        return result;
    }
    public static void main(String[] args) {
        int num1[]={5, 4, 3};
        int num2[]={1, 2, 4};
        int result[]=sumArray(num1, num2);
        printArray(result);
    }
}
```

# Multidimensional Arrays

- Thus far, you have used one-dimensional arrays to model linear collections of elements
- You can use a two-dimensional array to represent a matrix or a table

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

# Two-Dimensional Arrays

**// Declare array ref var**

```
dataType[][] refVar;
```

**// Create array and assign its reference to variable**

```
refVar = new dataType[10][10];
```

**// Combine declaration and creation in one statement**

```
dataType[][] refVar = new dataType[10][10];
```

**// Alternative syntax**

```
dataType refVar[][] = new dataType[10][10];
```

# Declare and Create Two-Dimensional Arrays

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

// or

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

```
matrix[0][0] = 3;
```

```
for (int i = 0; i < matrix.length; i++)  
    for (int j = 0; j < matrix[i].length; j++)  
        matrix[i][j] = (int) (Math.random() * 1000);
```

```
double[][] x;
```

# Two-Dimensional Array Illustration

	0	1	2	3	4
0					
1					
2					
3					
4					

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

matrix.length? 5

matrix[0].length? 5

	0	1	2	3	4
0					
1					
2		7			
3					
4					

```
matrix[2][1] = 7;
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

```
int[][] array = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9},  
    {10, 11, 12}  
};
```

array.length? 4

array[0].length? 3

# Declare, Create, and Initialize Using Shorthand Notations

- You can also use an array initializer to declare, create and initialize a two-dimensional array. For example,

```
int[][] array  
    = {  
        {1, 2, 3},  
        {4, 5, 6},  
        {7, 8, 9},  
        {10, 11,  
          12}  
    };
```

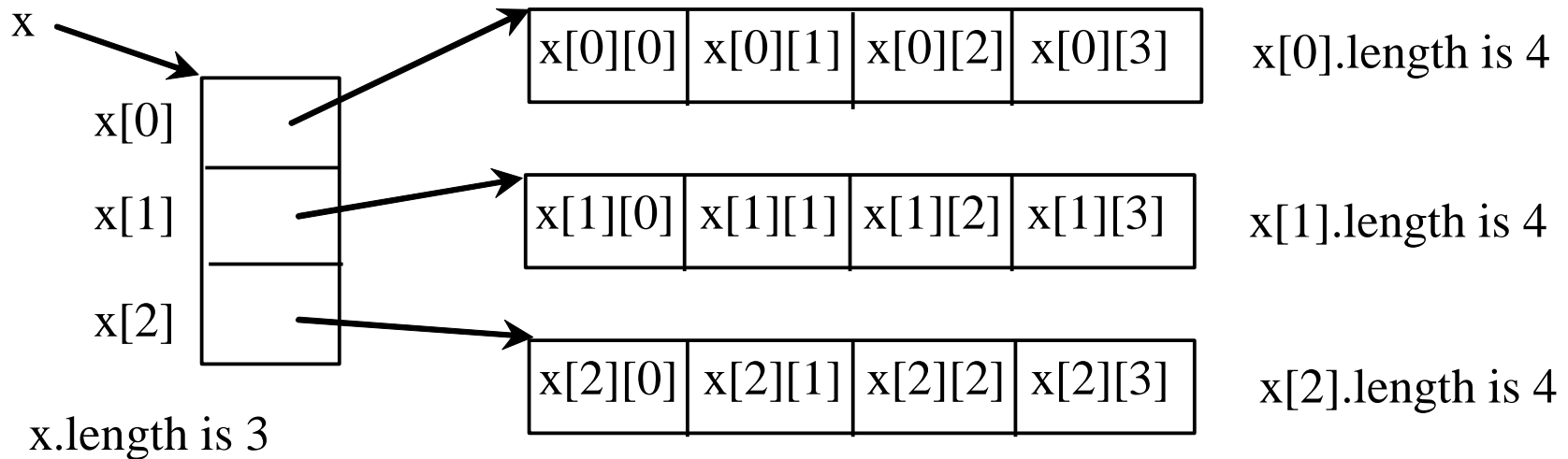
Same as

```
int[][] array = new int[4][3];  
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;  
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;  
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;  
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```



# Lengths of Two-Dimensional Arrays

- `int[][] x = new int[3][4];`



# Lengths of Two-Dimensional Arrays

```
int[][] array = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9},  
    {10, 11, 12}  
};  
  
array.length  
array[0].length  
array[1].length  
array[2].length  
array[3].length
```

`array[4].length`      `ArrayIndexOutOfBoundsException`

# Ragged Arrays

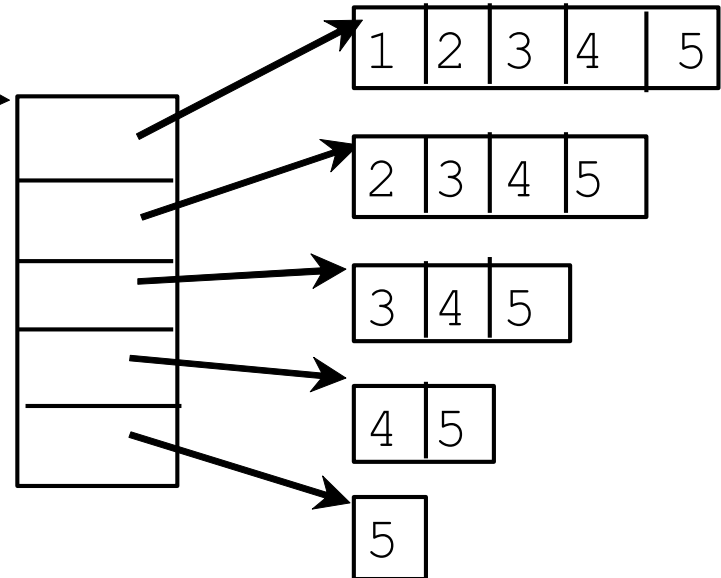
- Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as a *ragged array*. For example,

```
int[][] matrix = {  
    {1, 2, 3, 4, 5},  
    {2, 3, 4, 5},  
    {3, 4, 5},  
    {4, 5},  
    {5}  
};
```

```
matrix.length is 5  
matrix[0].length is 5  
matrix[1].length is 4  
matrix[2].length is 3  
matrix[3].length is 2  
matrix[4].length is 1
```

# Ragged Arrays

```
int[][] triangleArray = {  
    {1, 2, 3, 4, 5},  
    {2, 3, 4, 5},  
    {3, 4, 5},  
    {4, 5},  
    {5}  
};
```



# Multidimensional Arrays

- Occasionally, you will need to represent n-dimensional data structures. In Java, you can create n-dimensional arrays for any integer n.
- The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare n-dimensional array variables and create n-dimensional arrays for  $n \geq 3$ . For example, the following syntax declares a three-dimensional array variable `scores`, creates an array, and assigns its reference to `scores`

```
double[][][] scores = new double[10][5][2];
```