

Chapter 4: Arrays

TAO Yida

taoyd@sustech.edu.cn

Why using Array?

- ▶ Suppose that you need to read 100 numbers and find out how many numbers are above the average.
- ▶ Your program should compare each number with the average to determine whether it is above the average.

Declaring individual variables for each number, such as number0, number1, . . . , and number99 would be impractical

Why using Array?

- ▶ Java and most other high-level languages provide a data structure, the **array**, which stores a **fixed-size** sequential collection of elements of the **same type**.
- ▶ In the example, you can store all 100 numbers into an array and access them through a single array variable.

40	55	63	17	22	68	89	97	89
----	----	----	----	----	----	----	----	----

Objectives

- ▶ Declare and initialize arrays
- ▶ Access individual elements of arrays
- ▶ Use the enhanced `foreach` statement to process arrays
- ▶ Copying arrays
- ▶ 2D arrays & multidimensional arrays

Arrays

- ▶ An **array** (a widely-used data structure) is **a group of elements** containing values of **the same type**.
- ▶ **Arrays are objects**, so they're considered **reference types** (aka non-primitive types) (we will talk about this more later)

Declaring Arrays

- ▶ To use an array in a program, you must declare a variable to reference the array and specify the array's **element type**.

```
ElementType[] variableName;
```

- ▶ The ElementType can be any data type (primitive or reference type), and all elements in the array will have the **same data type**.

```
int[] intArray;
```

```
double[] doubleArray;
```

```
String[] stringArray;
```

Creating Arrays

- ▶ The declaration of an array variable does not allocate any space in memory for the array, and we cannot use the array before creating(initializing) it.

```
int[] c = new int[12];
```

- ▶ Like other objects (recall the usage of Scanner), arrays are created with the keyword **new**.
- ▶ **12** means the size of the array. When space for an array is allocated, the array size must be given.
- ▶ The size of an array **cannot be changed** after the array is created.

Creating Arrays

- ▶ The declaration of an array variable does not allocate any space in memory for the array, and we cannot use the array before creating(initializing) it.

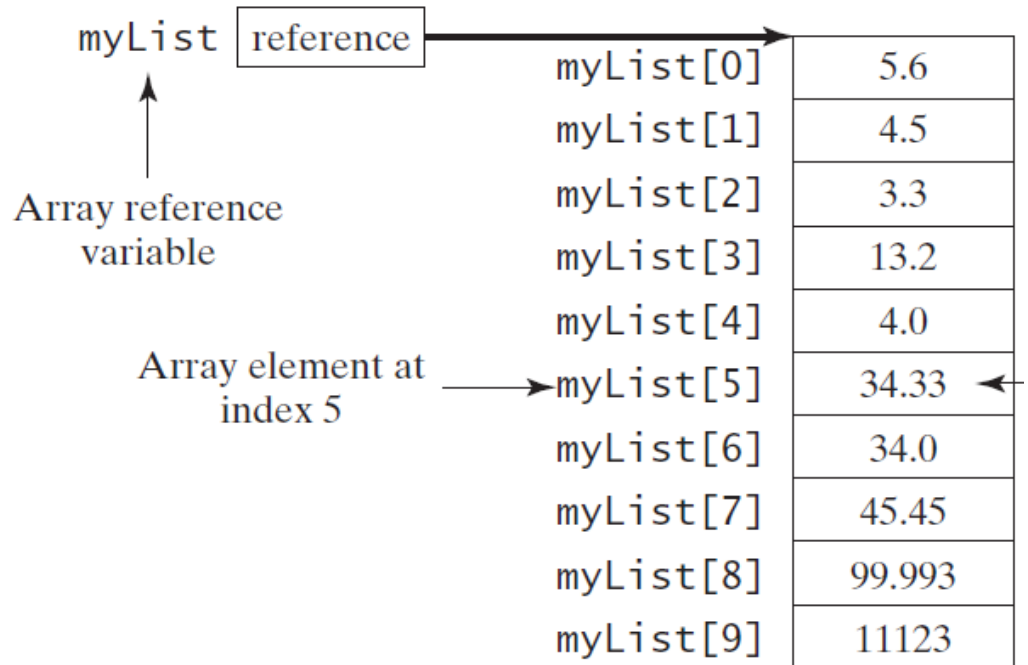
```
int[] c = new int[12];
```

- ▶ Variable `c` refers to an array of size 12 with elements of `int` type
- ▶ When an array is created, its elements are assigned **the default value** of `0` for the numeric primitive data types, `\u0000` for char types, and `false` for boolean types.

Accessing Array Elements

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

```
myList[0] = 5.6;  
myList[1] = 4.5;  
myList[2] = 3.3;  
myList[3] = 13.2;  
myList[4] = 4.0;  
myList[5] = 34.33;  
myList[6] = 34.0;  
myList[7] = 45.45;  
myList[8] = 99.993;  
myList[9] = 11123;
```



Size can be obtained using `arrayRefVar.length`.
For example, `myList.length` is 10.

Accessing Array Elements

myList[0]	5.6
myList[1]	4.5
myList[2]	3.3
myList[3]	13.2
myList[4]	4.0
myList[5]	34.33
myList[6]	34.0
myList[7]	45.45
myList[8]	99.993
myList[9]	11123

- ▶ The array elements are accessed through the index.
- ▶ The first element in every array has **index 0**.
- ▶ The highest index in an array is **the number of elements – 1**, i.e., `myList.length-1`

Accessing Array Elements

myList[0]	5.6
myList[1]	4.5
myList[2]	3.3
myList[3]	13.2
myList[4]	4.0
myList[5]	34.33
myList[6]	34.0
myList[7]	45.45
myList[8]	99.993
myList[9]	11123

`myList[5]` refers to the 6th element

- `myList` is the reference to the array (or name of the array for simplicity)
- `5` is the position number of the element (`index` or `subscript`)

Accessing Array Elements

myList[0]	5.6
myList[1]	4.5
myList[2]	3.3
myList[3]	13.2
myList[4]	4.0
myList[5]	34.33
myList[6]	34.0
myList[7]	45.45
myList[8]	99.993
myList[9]	11123

- ▶ A program can use an expression as an index (`c[1+a]`)
- ▶ An index must be a **nonnegative** integer (`c[-2]` causes error).
- ▶ If an `index < 0` or `index > array.length - 1`, you'll get an `ArrayIndexOutOfBoundsException` Exception

Accessing Array Elements

myList[0]	5.6
myList[1]	4.5
myList[2]	3.3
myList[3]	13.2
myList[4]	4.0
myList[5]	34.33
myList[6]	34.0
myList[7]	45.45
myList[8]	99.993
myList[9]	11123

- ▶ Array-access expressions can be used to get element value or on the left side of an assignment to place a new value into an array element

```
myList[1] = 2.2;  
System.out.println(myList[1]);
```

Print an array

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

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

[I@776ec8df

Array is a reference type. We cannot directly print a variable of the array type as we do for primitive types*

*A char array can be directly printed

Print an array

The int elements by default get the value of 0

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


```
System.out.printf("%s%8s\n", "Index", "Value");
```

```
// Using loop to output each array element's value
```

```
for(int counter = 0; counter < array.length; counter++) {  
    System.out.printf("%3d%8d\n", counter, array[counter]);  
}
```

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

Make sure the index is within **[0, array.length – 1]**

Otherwise: [java.lang.ArrayIndexOutOfBoundsException](#) 

Array Initialization

- ▶ You can create an array and initialize its elements with an **array initializer**—a comma-separated list of expressions enclosed in braces.

```
int[] n = new int[] { 10, 20, 30, 40, 50 };
```

- ▶ Compiler **counts the # of values in the list to determine the size of the array, then sets up the appropriate new operation “behind the scenes”**.
- ▶ Element `n[0]` is initialized to 10, `n[1]` is initialized to 20, and so on.

Array Initialization

- ▶ You can create an array and initialize its elements with an **array initializer**—a comma-separated list of expressions enclosed in braces.

```
int[] n = { 10, 20, 30, 40, 50 };
```

- ▶ Shortcut: initialize the array without using the **new** keyword
- ▶ This shortcut is allowed **only at the time of array declaration**

```
int[] array;  
array = {10, 20, 30, 40, 50};
```

Array initializer is not allowed here

Add 'new int[]' Alt+Shift+Enter

Array Initialization

How to initialize the array
to output these numbers?

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

```
System.out.printf("%s%8s\n", "Index", "Value");  
// output each array element's value  
for(int counter = 0; counter < array.length; counter++) {  
    System.out.printf("%3d%8d\n", counter, array[counter]);  
}
```

Index	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

Array Initialization

```
int[] array = {2,4,6,8,10,12,14,16,18,20};
```

```
System.out.printf("%s%8s\n", "Index", "Value");  
// output each array element's value  
for(int counter = 0; counter < array.length; counter++) {  
    System.out.printf("%3d%8d\n", counter, array[counter]);  
}
```

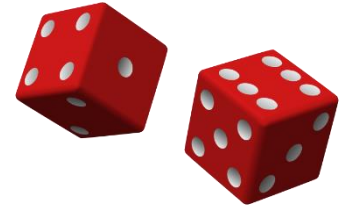
Index	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

Array Initialization

```
int[] array = new int[10];  
//calculate value for each array element  
for(int counter = 0; counter < array.length; counter++) {  
    array[counter] = 2 + 2 * counter;  
}  
System.out.printf("%s%8s\n", "Index", "Value");  
// output each array element's value  
for(int counter = 0; counter < array.length; counter++) {  
    System.out.printf("%3d%8d\n", counter, array[counter]);  
}
```

Index	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

A Dice-Rolling Program



- ▶ Suppose we want to roll a dice 6000 times and count the frequency of each side
- ▶ We can use separate counters as below
 - `int faceOneFreq, faceTwoFreq, ...`
- ▶ Now we have learned arrays. Is there a better design?



```
import java.util.Random;

public class DiceRolling {

    public static void main(String[] args) {
        Random generator = new Random();
        int[] frequency = new int[6];
        // roll 6000 times; use dice value as frequency index
        for(int roll = 1; roll <= 6000; roll++) {
            int face = generator.nextInt(6);
            frequency[face]++;
        }
        System.out.printf("%s%10s\n", "Face", "Frequency");
        // output the frequency of each face
        for(int face = 0; face < frequency.length; face++) {
            System.out.printf("%4d%10d\n", face+1, frequency[face]);
        }
    }
}
```

Use an array to track frequency

nextInt(6) generates [0, 5]

Execution Result

Face Frequency

1 1016

2 991

3 981

4 1011

5 988

6 1013

Objectives

- ▶ Declare and initialize arrays
- ▶ Access individual elements of arrays
- ▶ Use the enhanced **foreach** statement to process arrays
- ▶ Copying arrays
- ▶ 2D arrays & multidimensional arrays

foreach Statement

- ▶ Java supports a convenient for loop, known as a **foreach** loop, which enables you to traverse the array sequentially **without using an index variable**.

```
for (double e: myList) {  
    System.out.println(e);  
}
```

Avoid the possibility of “stepping outside” the array.

foreach Statement

- *arrayName* is the array through which to iterate.
- Elements in the array is named as *identifier*
- *ElementType* must be consistent with the type of the elements in the array.

```
for ( ElementType identifier : arrayName ) {  
    // do something with the identifier  
}
```

foreach Statement

- ▶ Simple syntax compared to the normal for statement

```
for ( int num : numbers ) {  
    // statements using num  
}
```

Semantically equivalent

```
for ( int i = 0; i < numbers.length; i++ ) {  
    int num = numbers[i];  
    // statements using num  
}
```

foreach Statement

- ▶ Often used to replace counter-controlled for statement when the code requires only read access to element values.

```
for ( int i = 0; i < numbers.length; i++ ) {  
    total += numbers[i];  
}
```



```
for ( int num : numbers ) {  
    total += num;  
}
```

Simpler and elegant
😊

foreach Statement

- ▶ Cannot be used to modify element values

```
for ( int num : numbers ) {  
    num = 0;  
}
```

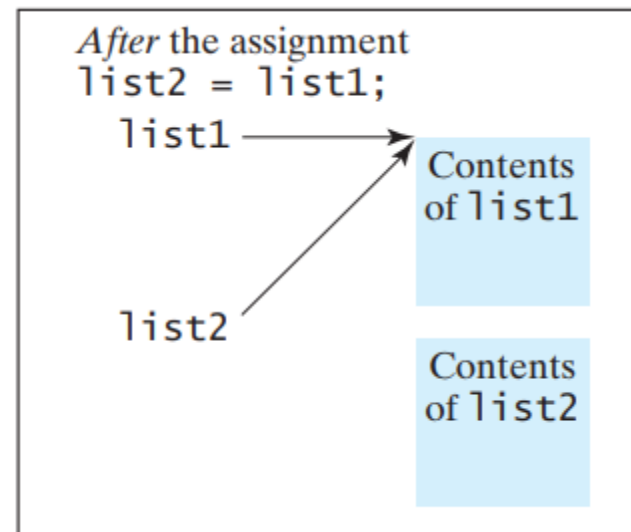
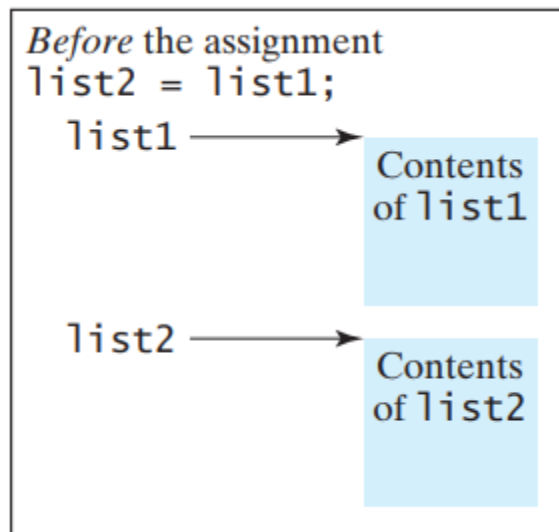
Can this change the array element values?
No! Only change the value of **num**

```
for ( int i = 0; i < numbers.length; i++ ) {  
    int num = numbers[i];  
    num = 0;  
}
```

Local variable **num** stores a copy of
the array element value

Copying Arrays

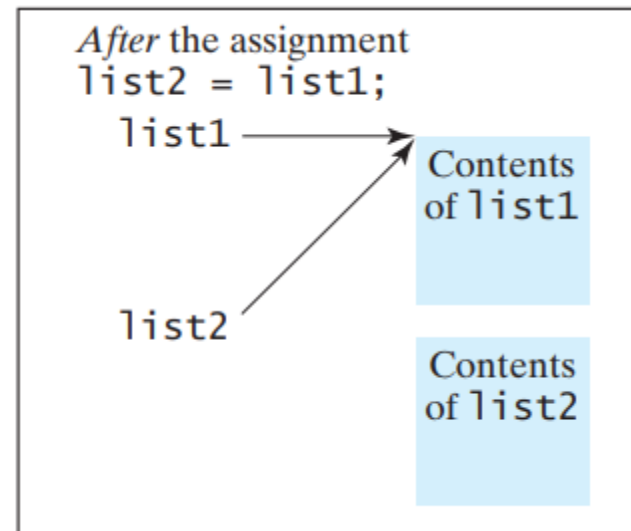
- ▶ The assignment statement **does not** copy the contents of the array referenced by `list1` to `list2`, but instead merely **copies the reference** value from `list1` to `list2`.
- ▶ After this statement, `list1` and `list2` reference the same array



Copying Arrays

- ▶ The assignment statement **does not** copy the contents of the array referenced by `list1` to `list2`, but instead merely **copies the reference** value from `list1` to `list2`.
- ▶ After this statement, `list1` and `list2` reference the same array

```
int[] list1 = {1,2,3,4,5};  
int[] list2 = {6,7,8,9};  
  
list2 = list1;  
System.out.println(Arrays.toString(list2));  
// [1, 2, 3, 4, 5]  
  
list1[3] = 100;  
System.out.println(Arrays.toString(list2));  
// [1, 2, 3, 100, 5]
```



Copying Arrays

- ▶ You can write a loop to copy every element from the source array to the corresponding element in the target array.

```
int[] sourceArray = {2, 3, 1, 5, 10};  
int[] targetArray = new int[sourceArray.length];  
for (int i = 0; i < sourceArray.length; i++) {  
    targetArray[i] = sourceArray[i];  
}
```


Objectives

- ▶ Declare and initialize arrays
- ▶ Access individual elements of arrays
- ▶ Use the enhanced `foreach` statement to process arrays
- ▶ Copying arrays
- ▶ 2D arrays & multidimensional arrays

One-Dimensional Arrays

- ▶ Arrays that we have considered up to now are **one-dimensional arrays**: a single line of elements.

	78	-9	520	0	14
Index	0	1	2	3	4

Example: an array of five random numbers

Two-Dimensional Arrays

- ▶ Data in real life often come in the form of a table

	Test 1	Test 2	Test 3	Test 4	Test 5
Student 1	87	96	70	68	92
Student 2	85	75	83	81	52
Student 3	69	77	96	89	72
Student 4	78	79	82	85	83

Example:
a gradebook

The table can be represented using a two-dimensional array in Java

Two-Dimensional (2D) Arrays

- 2D arrays are indexed by two subscripts: one for the **row number**, the other for the **column number**. Subscripts **start with 0**.

	Test 1	Test 2	Test 3	Test 4	Test 5
Student 1	87	96	70	68	92
Student 2	85	75	83	81	52
Student 3	69	77	96	89	72
Student 4	78	79	82	85	83

row column
gradebook[1][2]
(gradebook: name of array)

2D Array Basics (Similar to 1D Array)

- ▶ Similar to 1D array, each element in a 2D array should be of the same type: either primitive type or reference type
- ▶ Array access expression (subscripted variables) can be used just like a normal variable: `gradebook[1][2] = 77;`
- ▶ Array indices (subscripts) must be of type `int`, can be a literal, a variable, or an expression: `gradebook[1][j]`, `gradebook[i+1][j+1]`
- ▶ If an array index does not exist, JVM will throw an exception `ArrayIndexOutOfBoundsException`

Declaring and Creating 2D Arrays

- ▶ Declares a variable that references a 2D array of `int`

```
int[][] gradebook;
```

- ▶ Creates a 2D array (**50-by-6 array**) with **50 rows** (for 50 students) and **6 columns** (for 6 tests) and assign the reference to the new array to the variable `gradebook`

```
gradebook = new int[50][6];
```

Shortcut: `int[][] gradebook = new int[50][6];`

Array Initialization

We can initialize a 2D array by assigning to each element, or with nested array initializers

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

	[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}  
};
```

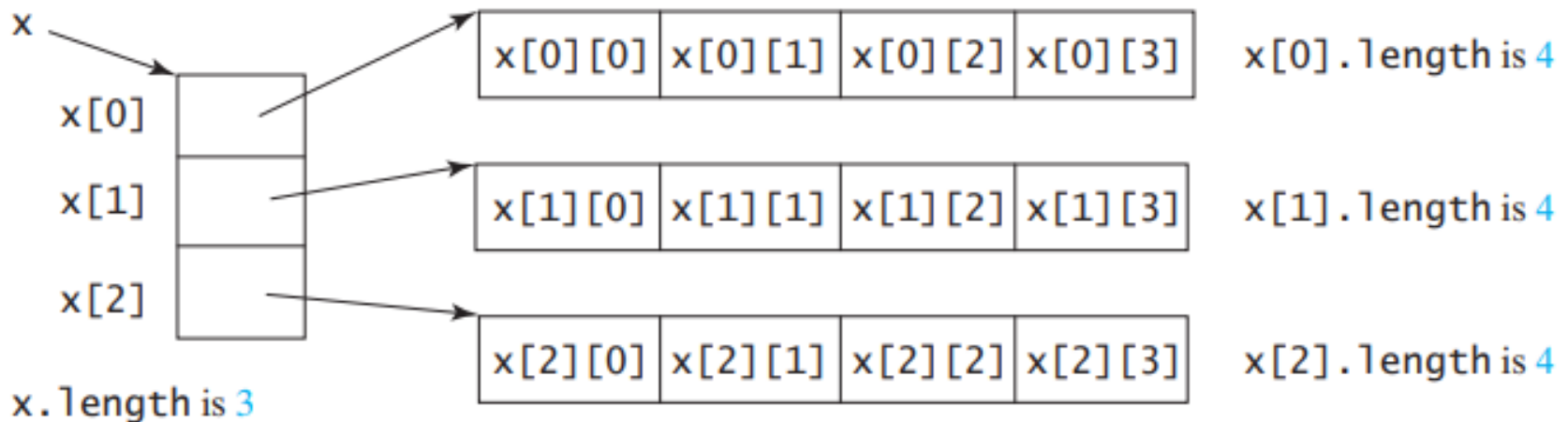
Equivalent

```
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 2D Arrays

A 2D array is actually an array in which each element is a 1D array

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

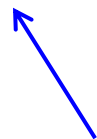


Ragged Arrays

- ▶ In 2D arrays, rows can have different lengths (ragged arrays)

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

1	2	3	4
5	6		
7	8	9	
10			



Row 1



Row 2



Row 3



Row 4

Note that the compiler will “**smartly**” determine the number of rows and columns

Why do we need ragged arrays?

```
1 * 1 = 1
1 * 2 = 2  2 * 2 = 4
1 * 3 = 3  2 * 3 = 6  3 * 3 = 9
1 * 4 = 4  2 * 4 = 8  3 * 4 = 12  4 * 4 = 16
1 * 5 = 5  2 * 5 = 10  3 * 5 = 15  4 * 5 = 20  5 * 5 = 25
1 * 6 = 6  2 * 6 = 12  3 * 6 = 18  4 * 6 = 24  5 * 6 = 30  6 * 6 = 36
1 * 7 = 7  2 * 7 = 14  3 * 7 = 21  4 * 7 = 28  5 * 7 = 35  6 * 7 = 42  7 * 7 = 49
1 * 8 = 8  2 * 8 = 16  3 * 8 = 24  4 * 8 = 32  5 * 8 = 40  6 * 8 = 48  7 * 8 = 56  8 * 8 = 64
1 * 9 = 9  2 * 9 = 18  3 * 9 = 27  4 * 9 = 36  5 * 9 = 45  6 * 9 = 54  7 * 9 = 63  8 * 9 = 72  9 * 9 = 81
```

Document:

1. "Hello, how are you?"
2. "I love programming."
3. "Natural language processing is fascinating."

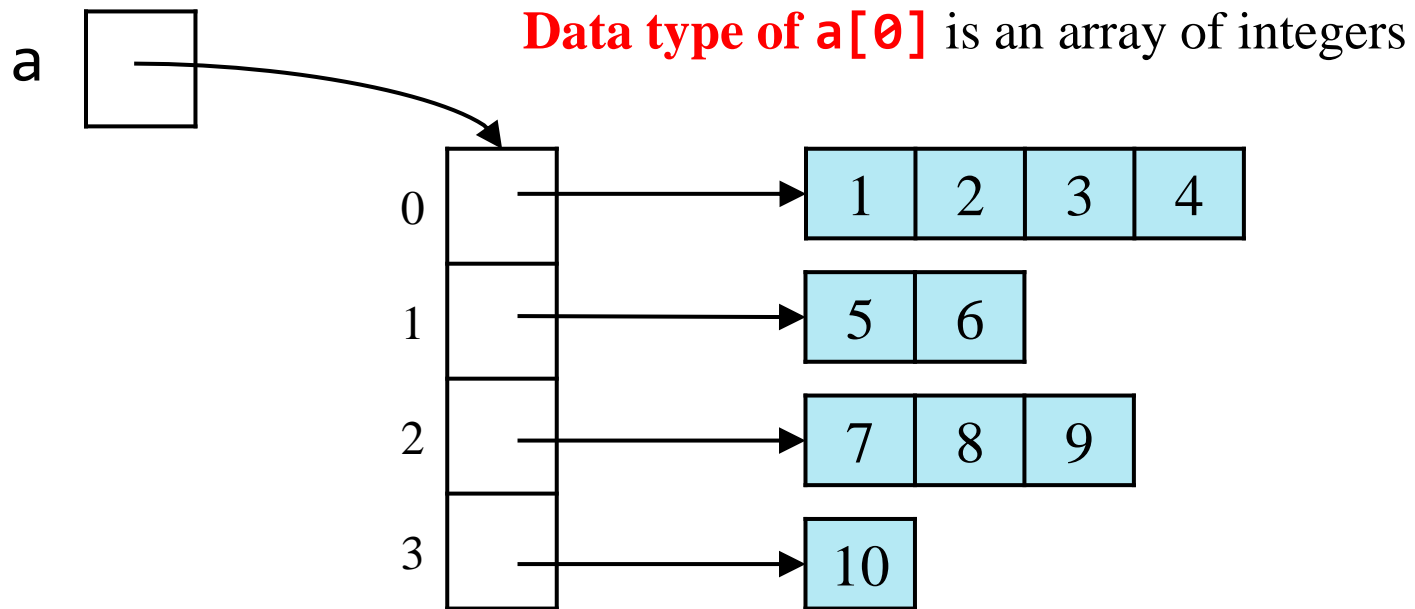
Ragged Array:

```
[["Hello", "how", "are", "you", "?"],  
 ["I", "love", "programming", "."],  
 ["Natural", "language", "processing", "is", "fascinating", "."]]
```

Ragged Arrays

- ▶ A 2D array is a 1D array of (references to) 1D arrays

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



Ragged Arrays

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

- ▶ What is the value of `a[0]`?
 - **Answer:** The reference (memory address) to the 1D array `{1, 2, 3, 4}`
- ▶ What is the value of `a.length`?
 - **Answer:** 4, the number of rows
- ▶ What the value of `a[1].length`?
 - **Answer:** 2, the second row only has 2 columns

Ragged Arrays

- ▶ Since a 2D array is a 1D array of (references to) 1D arrays, a 2D array in which each row has a different number of columns can also be created as follows:

```
int[][] b = new int[ 2 ][ ];    // create 2 rows
b[ 0 ] = new int[ 5 ]; // create 5 columns for row 0
b[ 1 ] = new int[ 3 ]; // create 3 columns for row 1

b[0][0] = 3;
b[1][2] = 4;
```

Ragged Arrays

- ▶ Since a 2D array is a 1D array of (references to) 1D arrays, a 2D array in which each row has a different number of columns can also be created as follows:

```
int[][] b = new int[ 3 ][ ];    // create 2 rows
b[ 0 ] = new int[]{ 1, 2, 3, 4 }; // initialize row 0
b[ 1 ] = new int[]{ 5, 6 };     // initialize row 1
b[ 2 ] = { 7, 8, 9 };          // compilation error!
```

Displaying 2D array

```
public static void main(String[] args) {  
    int[][] a = {{1, 2, 3, 4}, {5, 6}, {7, 8, 9}, {10}};  
  
}
```

```
1 2 3 4  
5 6  
7 8 9  
10
```

Displaying 2D array

```
public static void main(String[] args) {  
    int[][] a = {{1, 2, 3, 4}, {5, 6}, {7, 8, 9}, {10}};  
    // loop through rows  
    for(int row = 0; row < a.length; row++) {  
        // loop through columns  
        for(int column = 0; column < a[row].length; column++) {  
            System.out.printf("%d ", a[row][column]);  
        }  
        System.out.println();  
    }  
}
```

```
1 2 3 4  
5 6  
7 8 9  
10
```


Computing Average Scores for each student (using foreach statement)

```
public static void main(String[] args) {  
    int[][] gradebook = {  
        {87, 96, 70, 68, 92},  
        {85, 75, 83, 81, 52},  
        {69, 77, 96, 89, 72},  
        {78, 79, 82, 85, 83}  
    };  
}
```

```
82.6  
75.2  
80.6  
81.4
```

Computing Average Scores for each student (using foreach statement)

```
public static void main(String[] args) {  
    int[][] gradebook = {  
        {87, 96, 70, 68, 92},  
        {85, 75, 83, 81, 52},  
        {69, 77, 96, 89, 72},  
        {78, 79, 82, 85, 83}  
    };  
    for(grades : gradebook) {  
        int sum = 0;  
  
        System.out.printf("%.1f\n", ((double) sum)/grades.length);  
    }  
}
```

```
82.6  
75.2  
80.6  
81.4
```

Computing Average Scores for each student (using foreach statement)

```
public static void main(String[] args) {  
    int[][] gradebook = {  
        {87, 96, 70, 68, 92},  
        {85, 75, 83, 81, 52},  
        {69, 77, 96, 89, 72},  
        {78, 79, 82, 85, 83}  
    };  
    for(int[] grades : gradebook) {  
        int sum = 0;  
        for(int grade : grades) {  
            sum += grade;  
        }  
        System.out.printf("%.1f\n", ((double) sum)/grades.length);  
    }  
}
```

82.6
75.2
80.6
81.4

Can we move int sum=0 before the for loop?

What is the value of matrix?

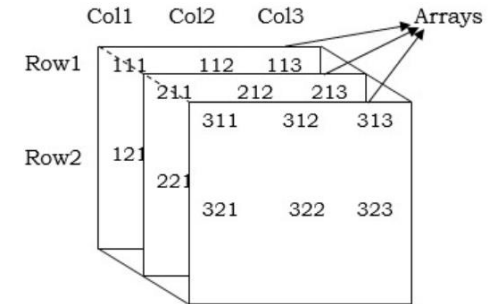
```
int[][] matrix = new int[4][];  
matrix[0] = new int[]{1,2,3};  
matrix[1] = new int[]{1,2,3};  
matrix[2] = new int[]{1,2,3};  
matrix[3] = new int[]{1,2,3};
```

```
for(int col=0;col<3;col++){  
    for(int row=1;row<4;row++){  
        matrix[row][col] += matrix[row-1][col];  
    }  
}
```

1	2	3
1	2	3
1	2	3
1	2	3

Multidimensional Arrays

- ▶ Arrays can have more than two dimensions.
 - `int[][][] a = new int[3][4][5];`
- ▶ Concepts for multidimensional arrays (2D above) can be generalized from 2D arrays
 - 3D array is an 1D array of (references to) 2D arrays, each of which is a 1D array of (references to) 1D arrays
- ▶ 1D array and 2D arrays are most commonly-used.



Multidimensional Arrays



An RGB image of m rows and n columns is stored as an $3 \times m \times n$ data array that defines red, green, and blue color components for each individual pixel

Multidimensional Arrays

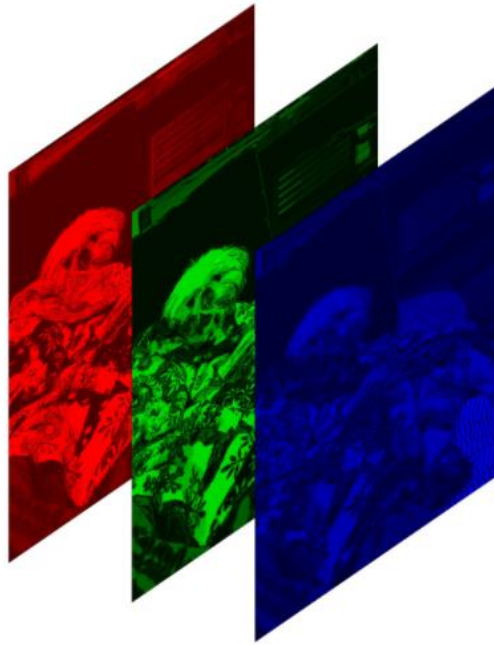


Image credit: Diane Rohrer

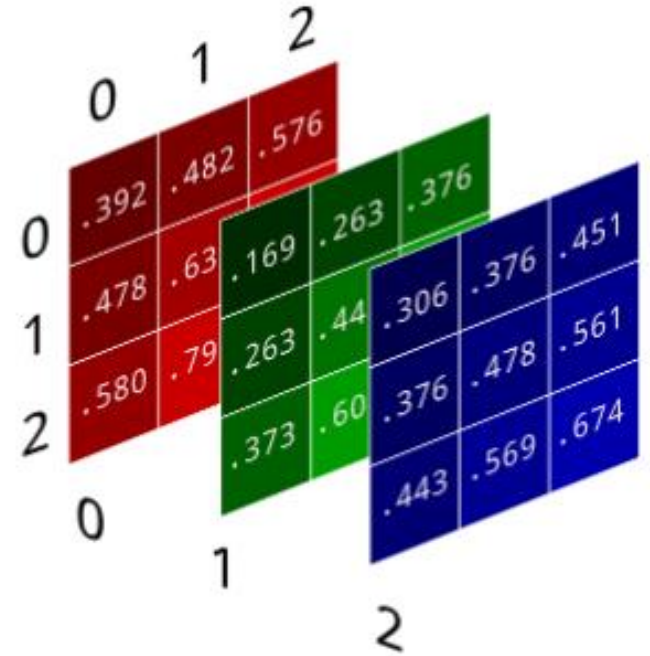


Image: <https://www.kdnuggets.com/2019/12/convert-rgb-image-grayscale.html>