

# 303.3.2 - Java Arrays

# Lesson 1

## Java Arrays

### Learning Objectives:

- ❑ In this presentation, we will deep-dive into the core concept of the Java language, which is referred to as Arrays.
- ❑ By the end of this presentation, learners will be able to:
  - Define the definition of Arrays, including dynamic arrays.
  - Demonstrate how to declare the array, and initialize the elements in an array
  - Define how to access elements of an Array in Java and Iterate over an Array using Loop.



# Outline

3

- ❑ Topic 1a: Overview of an Array.
- ❑ Topic 1b: Declaring and Initialize Arrays in Java.
- ❑ Topic 1c: Access elements of an Array in Java.
- ❑ Using Indexed Variables.
- ❑ Topic 1d: The length of an Array.
- ❑ Topic 2: Iterating over an Array using Loop.
- ❑ Topic 2a: Traverse Through Array Using for Loop
- ❑ Topic 2b: Array Access With an Enhance - *for* Loop.
- ❑ Topic 3a: Overview of Multidimensional Arrays.
- ❑ Topic 3b: Our First Array: `main(String[] args)`.
- ❑ Applications of Arrays.

# Lesson 1

4

## Introduction to Arrays

In this lesson, we will begin looking at what an Array is, and how to create or declare an Array in Java.



# Opening Problem

5

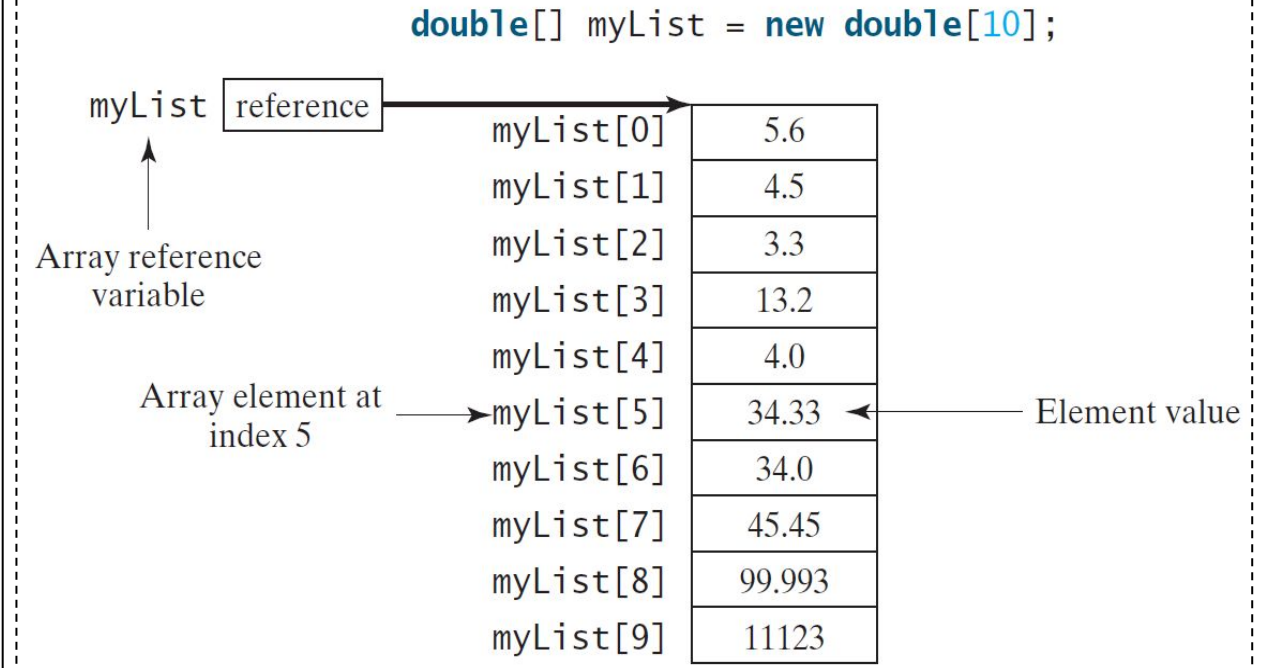
1. Suppose you want to find the average of the grades for a class of 30 students. You certainly do not want to create 30 variables, such as mark1, mark2, ..., mark30.
  - a. **Solution:** Use a single variable, called an **Array**, with 30 elements (or items).



# Topic 1a: Overview of Array

6

- ❑ The Array is a data structure, which stores a **fixed-size** sequential collection of elements of the same type. A **Java Array length cannot be changed**.
- ❑ An Array is used to store a collection of **data** or a collection of similar types of **elements**, which have contiguous memory location.
- ❑ An Array in Java is an **object** because the direct **Superclass** of an Array is **Object Class**. Since Arrays are objects, all methods of Object Class may be invoked on an Array.
- ❑ All Arrays are **reference types** no matter their element type. Memory for Arrays is always allocated on the heap.
- ❑ In Java, an Array can hold both **primitives values** and **objects type values**.



Elements start with an index of zero and last index is length - 1.

The size of an Array is determined at compile time.

# Topic 1b: Declare and Initialize Arrays in Java

7

Declare and Initialize Arrays in Java by:

- ❑ Declaration without initializing elements/values.
- ❑ Declaration using new operator.
- ❑ Declaration and initialization in one step.

# Declaration Without Initializing Elements/Value

8

1. Syntax → `datatype[] arrayName;`

Example:

- ❑ `double[] arrayName;`
- ❑ `String[] arrayName;`
- ❑ `int[] arrayName;`

2. Syntax → `datatype arrayName[];` // This style is allowed, but not preferred.

Example:

- ❑ `double myList[];`



# Declaration Using New Operator

9

❑ Syntax → `datatype[] arrayName = new datatype[arraySize];`

❑ Example:

```
double[] myList = new double[10]; //allocates memory for myList.
```

```
myList[0] //references the first element in the array.
```

```
myList[9] //references the last element in the array.
```

Instead of hard-coding for the last element, we could use the array's *length*. It is useful when the last index is unknown:

```
myList[myList.length - 1] //references the last element in the array.
```

# Example: Declaration Using New Operator

10

```
public static void main(String args[]) {  
    int month_days[];  
    month_days = new int[12]; // declaring and creating array with size  
    month_days[0] = 31; // initializing values  
    month_days[1] = 28; // initializing values  
    month_days[2] = 31; // initializing values  
    month_days[3] = 30; // initializing values  
    month_days[4] = 31; // initializing values  
    month_days[5] = 30; // initializing values  
    month_days[6] = 31; // initializing values  
    month_days[8] = 30; // initializing values  
    month_days[9] = 31; // initializing values  
    month_days[10] = 30; // initializing values  
    month_days[11] = 31; // initializing values  
    System.out.println("April has " + month_days[3] + " days.");  
}
```

In Arrays, we can access the specific element by its index within square brackets.

## Output:

April has 30 days.

# Array Declaration and Initialization in One Step

11

We can also declare and initialize arrays in a single line without using a new operator, as shown below:

**Syntax** → `dataType[] arrayName = { comma separated values };`

**Example** → `double[] myList = {1.9, 2.9, 3.4, 3.5};`

The above statement is equivalent to the following statements:

```
double[] myList = new double[4];
```

```
myList[0] = 1.9;
```

```
myList[1] = 2.9;
```

```
myList[2] = 3.4;
```

```
myList[3] = 3.5;
```

# Array Declaration and Initialization in One Step

(continued)

12

Remember: Array declaration and initialization must be done in a single statement. For example, the following statement is **incorrect** and **will not** compile:

```
double[] myArray;  
myArray = {1.9, 2.9, 3.4, 3.5};
```

# Example: Array Declaration and Initialization in One Step

13

The following code creates an initialized array of integers:

```
public static void main(String args[]) {  
    int[] month_days = {31,28,31,30,31,30,31,30,31,30,31};  
    System.out.println("April has " + month_days[3] + "days.");  
}
```

## Output:

April has 30 days.

In Arrays, we can access the specific element by its index within square brackets.

# Topic 1c: Access Elements of an Array in Java

14

## Array Indexing:

- The array elements are accessed by **index**. Array indexing is 0-based (e.g., ranging from **0** to **arrayVar.length-1**):

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

*//myList holds 10 elements indexed from 0 to myList.length-1*

- Each element in the array can be accessed using the following syntax, known as an **indexing expression**, or simply **indexer**.

```
arrayVar[index]; // index is an integer
```



# Using Indexed Variables

15

- ❑ After an array is created, its indexed variables can be used in the same way as regular variables. The following statement adds the first two values in an array and assigns the result to the third index in the array:

```
myList[2] = myList[0] + myList[1];
```

- ❑ Be careful not to attempt to access values past the end of the array:

```
double val = myList[10]; // Causes an exception!!
```

- ❑ This results in an [ArrayIndexOutOfBoundsException](#).

# Topic 1d: The Length of an Array

16

- ❑ Once an Array is created, its size is fixed, and cannot be changed.
- ❑ An array's size is given by its *length* property:

```
int size = arrayRefVar.length;
```

- ❑ For example: **myList.length;** // returns 10

# Example - The Length of an Array

17

```
public class Test {  
    public static void main(String[] args)  
    {  
        // Here str is the array name of String type.  
        String[] str = { "Java", "FOR", "EVERYONE" };  
        System.out.println(str.length);  
    }  
}
```

**Explanation:** Here, the str is an array of type string, and that is why str.length is used to find its length.

**Output:**

3

## Iterating Over an Array Using Loop

In this lesson, we will discuss Traverse through an array.



# Topic 2a: Traverse Through Array Using *for* Loop

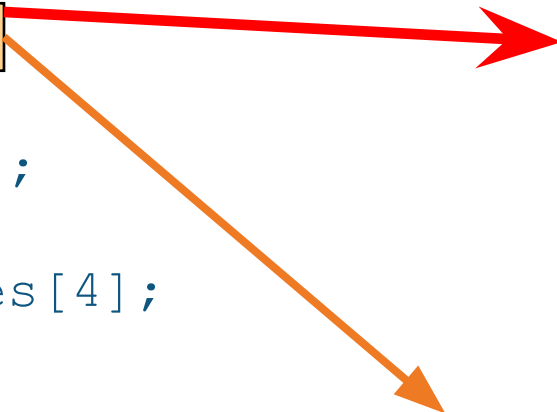
19

## Array Access With a *for* Loop - 1:

In Java, we can also loop through each element of the array.

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

After the array is created



0	0
1	0
2	0
3	0
4	0

Declare an array variable named *values*.  
Create the array using the *new* operator.  
Assign the result of *new* to *values*.

# Array Access With a *for* Loop - 2

20

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

i is initialized to 1.

After the array is created

0	0
1	0
2	0
3	0
4	0

Arrays are often used with for loops. The loop counter becomes the Array indexer.



# Array Access With a *for* Loop - 3

21

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

i (=1) is less than 5, so the looping begins.

After the array is created

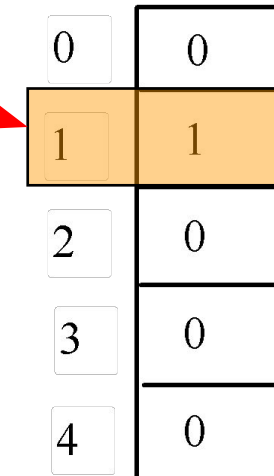
0	0
1	0
2	0
3	0
4	0

# Array Access With a *for* Loop - 4

22

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

After the first iteration



0	0
1	1
2	0
3	0
4	0

After this line executes, values[1] has been assigned 1.

# Array Access With a *for* Loop - 5

23

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

After i++, i has a value of 2.

After the first iteration

0	0
1	1
2	0
3	0
4	0

# Array Access With a *for* Loop - 6

24

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

i (= 2) is less than 5, so the looping continues.

After the first iteration

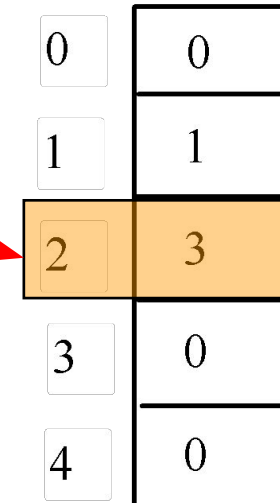
0	0
1	1
2	0
3	0
4	0

# Array Access With a *for* Loop - 7

25

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

After the second iteration



0	0
1	1
2	3
3	0
4	0

After this line executes,  
values[2] has been assigned 3.

# Array Access With a *for* Loop - 8

26

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

After i++, i has a value of 3.

After the second iteration

0	0
1	1
2	3
3	0
4	0



# Array Access With a *for* Loop - 9

27

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

i (=3) is still less than 5,  
so the looping continues.

After the second iteration

0	0
1	1
2	3
3	0
4	0

# Array Access with a *for* Loop - 10

28

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

After the third iteration

0	0
1	1
2	3
3	6
4	0

After this line executes,  
values[3] has been assigned 6.

# Array Access with a *for* Loop - 11

29

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

After the third iteration

0	0
1	1
2	3
3	6
4	0

After i++, i has a value of 4.

# Array Access With a *for* Loop - 12

30

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

i (=4) is still less than 5,  
so the looping continues.

After the third iteration

0	0
1	1
2	3
3	6
4	0

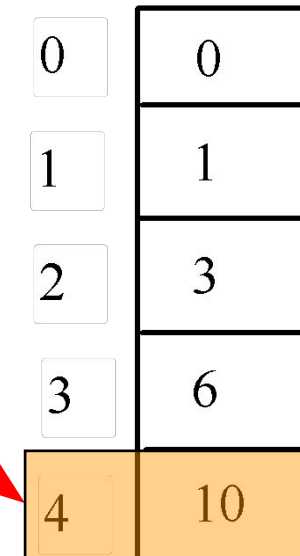
# Array Access With a *for* Loop - 13

31

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

After this line executes,  
values[4] has been assigned 10.

After the fourth iteration



0	0
1	1
2	3
3	6
4	10

# Array Access With a *for* Loop - 14

32

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

After i++, i has a value of 5.

After the fourth iteration

0	0
1	1
2	3
3	6
4	10



# Array Access With a *for* Loop - 15

33

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

$i < 5$  evaluates false. The loop exits.

After the fourth iteration

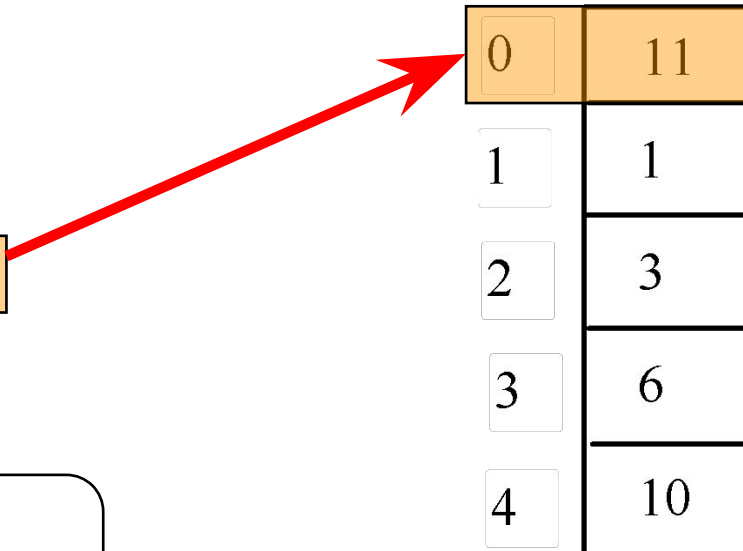
0	0
1	1
2	3
3	6
4	10

# Array Access With a *for* Loop - 16

34

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

After this line executes,  
values[0] has been assigned 11.



0	11
1	1
2	3
3	6
4	10

# Topic 2b: Array Access With an Enhance - *for* Loop

35

JDK 5 introduces a new loop syntax known as **enhanced for-loop (or for-each loop)** to facilitate processing of arrays and collections. It takes the following syntax:

## Syntax:

```
for(dataType item : array) {  
    ...  
}
```

Here:

- Array - an Array.
- Item - each item of array is assigned to this variable.
- DataType - the datatype of the Array.

## Example : Print Array Elements

```
public static void main(String[] args) {  
  
    // create an array  
    int[] numbers = {3, 9, 5, -5};  
  
    // for each loop  
    for (int number: numbers) {  
        System.out.println(number);  
    }  
}
```

# Hand-On Lab

36

Complete this [LAB- 3.3.3 -Array](#). You can find this lab on Canvas under the **Guided Lab section**.

If you have technical questions while performing the lab activity, ask your instructors for assistance.

## Overview of Multidimensional Arrays

In this lesson, we will discuss two-dimensional arrays, and how to declare and initialize the two-dimensional array.



# Topic 3a: Overview of Multidimensional Arrays

38

- ❑ In Java, we can declare an **array of arrays**, known as a **multidimensional array** or **two-dimensional array**. In a two-dimensional array, the elements can be arranged in **rows** and **columns**.

- ❑ **Syntax:**

```
datatype[][] arrayName = new datatype[column][row];
```

- ❑ Illustration of a **6 x 5 array** (6 rows, 5 columns).

	A	B	C	D	E
0	10	12	43	11	22
1	20	45	56	1	33
2	30	67	32	14	44
3	40	12	87	14	55
4	50	86	66	13	66
5	60	53	44	12	11

	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

# 1 - Declaring a Multidimensional Array

39

Consider the following matrix, which has integer **elements** and consists of **six rows** and **five columns**.

	A	B	C	D	E
0	10	12	43	11	22
1	20	45	56	1	33
2	30	67	32	14	44
3	40	12	87	14	55
4	50	86	66	13	66
5	60	53	44	12	11

```
//Declaring array 6x5 array of int
```

```
int[][] matrix2d = new int[6][5];
```

```
// Initialize elements of array
```

```
matrix2d[0][0] = 10;    // index position 0A
```

```
matrix2d[0][1] = 12;    // index position 0B
```

```
matrix2d[0][2] = 43;    // index position 0C
```

```
matrix2d[0][3] = 11;    // index position 0D
```

```
matrix2d[0][4] = 22;    // index position 0E
```

The first row is row 0. The columns then go from 0 to 4, which is 5 items. Now, fill the 2<sup>nd</sup>, 3<sup>rd</sup>, 4<sup>th</sup>, 5<sup>th</sup> and 6<sup>th</sup> row on your own!



## 2 - Declaring and Initializing Multidimensional Arrays

40

In one step:

Consider the following matrix, which has integer **elements** and consists of **six rows** and **five columns**.

	A	B	C	D	E
0	10	12	43	11	22
1	20	45	56	1	33
2	30	67	32	14	44
3	40	12	87	14	55
4	50	86	66	13	66
5	60	53	44	12	11

You can also use the shorthand notation to create and initialize the array as follows:

```
int[][] matrix2d = {  
    {10,12,43,11,22},  
    {20,45,56,1,33},  
    {30,67,32,14,44},  
    {40,12,87,14,55},  
    {50,86,66,13,66},  
    {60,53,44,12,11}  
};
```

# Iterating Over an Multidimensional Array

41

## Using Loop:

We can use nested loops to access all values in a multidimensional array.

```
public static void main(String[] args) {  
    // Suppose an array matrix is declared as follows:  
    int[][] matrix = new int[2][2];  
    // assign random values to the array using the following loop  
    for (int row = 0; row < matrix.length; row++) {  
        for (int column = 0; column < matrix[row].length; column++) {  
            matrix[row][column] = (int) (Math.random() * 1000);  
            System.out.print(matrix[row][column] + " ");  
        }  
    }  
}
```

## Output:

577 738 458 198

# Topic 3b: Our First Array: `main(String[] args)`

42

- Hopefully, you noticed that the `main()` method of each of our programs includes an Array parameter:

```
public static void main(String[] args) {  
    // do stuff with args  
}
```

- Our `main()` method is the entry point for the JVM, but it is also just a regular method that we can call from elsewhere in a program:

```
public class A {  
    public static void main(String[] args) {  
        String[] strings = {"New York",  
                            "Boston", "Atlanta"};  
        B.main(strings);  
    }  
}
```

```
class B {  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++)  
            System.out.println(args[i]);  
    }  
}
```

# Applications of Arrays

43

- Arrays are used as the building blocks to build other data structures such as array lists, heaps, hash tables, vectors, and matrices.
- Arrays are used for different sorting algorithms such as insertion sort, quick sort, bubble sort, and merge sort.

# Check Your Knowledge

44

Q1: Which is the correct syntax for **declaring/initializing** an array of integers?

- a. `[]int a = [10]int;`
- b. `int a[10];`
- c. `int[10] a = new int[10];`
- d. `int a[10] = new int[10];`
- e. `int[] a = new int[10];`

# Check Your Knowledge

45

Q 2: Which is the correct syntax for quickly **declaring/initializing** an array of integers to store a particular list of values?

- a. `int[] a = {17, -3, 42, 5, 9, 28};`
- b. `int a {17, -3, 42, 5, 9, 28};`
- c. `int[] a = new int[6] {17, -3, 42, 5, 9, 28};`
- d. `int[6] a = {17, -3, 42, 5, 9, 28};`
- e. `int[] a = new {17, -3, 42, 5, 9, 28} [6];`

# Check Your Knowledge

46

Q3: Fill in the array with the values that would be stored after the code executes:

```
int[] data = new int[8];  
data[0] = 3;  
data[7] = -18;  
data[4] = 5;  
data[1] = data[0];
```

**To answer, write the code and check the output.**

```
int x = data[4];  
data[4] = 6;  
data[x] = data[0] * data[1];
```

Index	0	1	2	3	4	5	6	7
Values								



# Check Your Knowledge

47

Q4: Fill in the array with the values that would be stored after the code executes:

```
int[] list = {2, 18, 6, -4, 5, 1};  
for (int i = 0; i < list.length; i++) {  
    list[i] = list[i] + (list[i] / list[0]);  
}
```

Index	0	1	2	3	4	5
Values						

To answer, write the code and check the output.

# Practice Assignment - Array

48

Complete the “**303.3.2 Practice Assignment -Array**” assignment for practice. You will find the assignment on Canvas, under the **Assignment** section.

Use your office hours to complete this assignment. If you have technical questions while performing the practice assignment, ask your instructors for assistance.

# Check Your Knowledge

49

Q1: Which of the following are limitations of Array?

- a) It is fixed in size.
- b) It allows us to store only the same types of elements.
- c) It does not provide built-in methods to perform different operations on the array.
- d) All of the above.

Q2: An Array can hold \_\_\_\_\_.

- a) Only primitive values.
- b) Only object-type values.
- c) Both primitive and object-type values.
- d) None of the above.

# Summary

50

- ❑ Java provides the data structure, an Array, which stores a fixed-sized, sequential collection of elements of the same type. An Array is used to store a collection of data, but it is often more useful to think of an Array as a collection of variables of the same type.
- ❑ Arrays can be passed as parameters to a function by reference only.
- ❑ An array is an ordered collection of elements of the same type and is identified by a pair of square brackets [ ]. To use an array, you need to:
  - Declare the array with a name and a type.
  - Use a plural name for an array (e.g., marks, rows, or numbers). All elements of an array belong to the same type.
  - Declare and Initialize Arrays in Java by:
    - ❑ Declaration without initializing elements/values.
    - ❑ Declaration using new operator.
    - ❑ Declaration and initialization in one step.

# References

51

- <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>
- <https://math.hws.edu/javanotes/c3/s8.html#:~:text=An%20array%20is%20a%20data,numbering%20always%20starts%20at%20zero.>
- <http://site.iugaza.edu.ps/mjdalloul/files/2016/09/Lab-8--Arrays-II.pdf>

# Questions?

52



# End of Module

53

