جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الانظمة الطبية الذكية

# Lecture: (1)

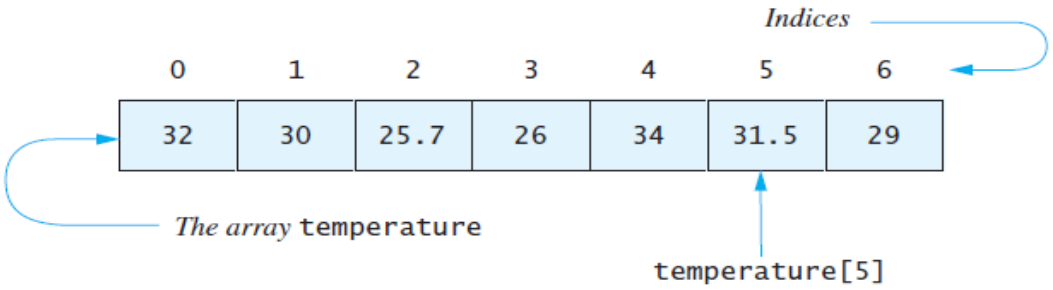
# Arrays Part I

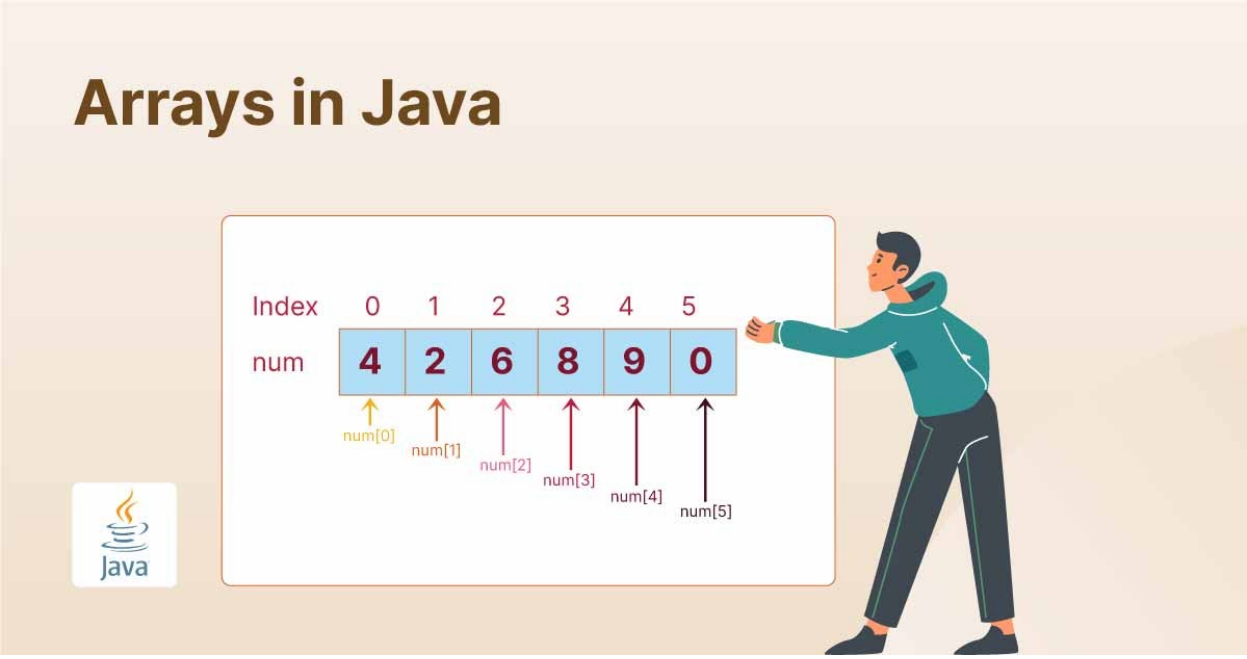
**Subject: Computer Programming II**
**Level: First**
**Lecturer: Dr. Maytham N. Meqdad**

# Arrays Part I

An array is a special kind of object used to store a collection of data. An array differs from the other objects you have seen in two ways:

- All the data stored in an array must be of the same type. For example, you might use an array to store a list of values of type `double` that record rainfall readings in centimeters. Or you might use an array to store a list of objects of some class called `Species` that contain the records for various endangered species.
- An array object has only a small number of predefined methods. Because arrays were used by programmers for many years before classes were invented, they use a special notation of their own to invoke those few predefined methods, and most people do not even think of them as methods.




Arrays in Java
Index 0 1 2 3 4 5
num 4 2 6 8 9 0
num[0]
num[1]
num[2]
num[3]
num[4]
num[5]
Java

# Why Arrays

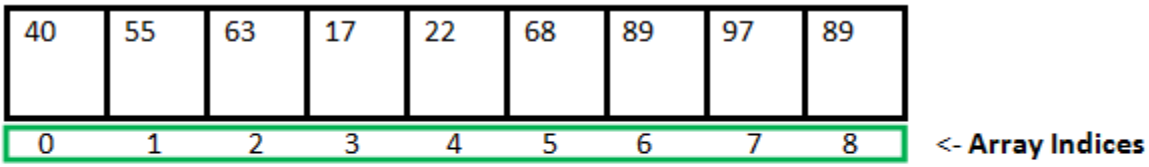
# Arrays in Java

**In Java, Array** is a group of like-typed variables referred to by a common name. Arrays in Java work differently than they do in C/C++. Following are some important points about Java arrays.

**Arrays in Java**

- In Java, all arrays are dynamically allocated. (discussed below)
- Arrays may be stored in contiguous memory [consecutive memory locations].
- Since arrays are objects in Java, we can find their length using the object property *length*. This is different from C/C++, where we find length using sizeof.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered, and each has an index beginning with 0.
- Java array can also be used as a static field, a local variable, or a method parameter.

An array can contain primitives (int, char, etc.) and object (or non-primitive) references of a class depending on the definition of the array. In the case of primitive data types, the actual values might be stored in contiguous memory locations(JVM does not guarantee this behavior). In the case of class objects, the actual objects are stored in a heap segment.




**Array Length = 9**
**First Index = 0**
**Last Index = 8**

# Creating, Initializing, and Accessing an Arrays

In Java, you can initialize an array in a few different ways depending on your requirements. Here are some common ways to initialize an array:

1. **Specify the size and then assign values individually:**

```
int[] myArray = new int[5];
myArray[0] = 1;
myArray[1] = 2;
myArray[2] = 3;
myArray[3] = 4;
myArray[4] = 5;
```

2. **Initialize with values in one line:**

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

3. **Initialize an array of a specified size with default values:**

```
int size = 5;
int[] myArray = new int[size]; // Initializes with default values (0 for int)
```

4. **Initialize an array with a specific value for all elements:**

```
int size = 5;
int value = 10;
int[] myArray = new int[size];
Arrays.fill(myArray, value);
```

Remember to import `java.util.Arrays` if you're using the `Arrays.fill()` method.

These are just a few examples. Depending on your specific use case, you might choose one over the other.

## One-Dimensional Arrays

The general form of a one-dimensional array declaration is

```
-- type var-name[];
-- type[] var-name;
```

An array **declaration** has two components: the type and the name. *type* declares the element type of the array. The element type determines the data type of each element that comprises the array. Like an array of integers, we can also create an array of other primitive data types like char, float, double, etc., or user-defined data types (objects of a class). Thus, the element type for the array determines what type of data the array will hold.

**Example:**

```
// both are valid declarations
int intArray[];
int[] intArray;
// similar to int we can declare
// byte , short, boolean, long, float
// double, char
// an array of references to objects of
// the class MyClass (a class created by user)
MyClass myClassArray[];
// array of Object
Object[]  ao,
// array of Collection
// of unknown type
Collection[] ca;
```

Although the first declaration establishes that int Array is an array variable, **no actual array exists**. It merely tells the compiler that this variable (int Array) will

hold an array of the integer type. To link int Array with an actual, physical array of integers, you must allocate one using **new** and assign it to int Array.

# Instantiating an Array in Java

When an array is declared, only a reference of an array is created. To create or give memory to the array, you create an array like this: The general form of *new* as it applies to one-dimensional arrays appears as follows:

```
var-name = new type [size];
```

Here, *type* specifies the type of data being allocated, *size* determines the number of elements in the array, and *var-name* is the name of the array variable that is linked to the array. To use *new* to allocate an array, **you must specify the type and number of elements to allocate.**

**Example:**

```
//declaring array
int intArray[];
// allocating memory to array
intArray = new int[20];
// combining both statements in one
int[] intArray = new int[20];
```

**Note:** The elements in the array allocated by *new* will automatically be initialized to **zero** (for numeric types), **false** (for boolean), or **null** (for reference types). Do refer to default array values in Java.

Obtaining an array is a two-step process. First, you must declare a variable of the desired array type. Second, you must allocate the memory to hold the array, using new, and assign it to the array variable. Thus, **in Java**, **all arrays are dynamically allocated.**

Printing Arrays Array java

# Printing Arrays java

Java array is a data structure where we can store the elements of the same data type. The elements of an array are stored in a contiguous memory location. So, we can store a fixed set of elements in an array.

There are following ways to print an array in Java:

- Java **for** loop
- Java **for-each** loop
- Java **Arrays.toString()** method
- Java **Arrays.deepToString()** method
- Java **Arrays.asList()** method
- Java **Iterator** Interface
- Java **Stream** API

Printing arrays in Java can be done using various methods. Here are some common approaches:

## 1. Using a loop:

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

for (int i = 0; i < myArray.length; i++) {
    System.out.print(myArray[i] + " ");
}
```

## 2. Using enhanced for loop (for-each loop):

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

for (int num : myArray) {
    System.out.print(num + " ");
}
```

## 3. Using `Arrays.toString()` method for simple arrays:

```
import java.util.Arrays;

int[] myArray = {1, 2, 3, 4, 5};
System.out.println(Arrays.toString(myArray));
```

4. **Using `Arrays.deepToString()` method for multidimensional arrays:**

```
import java.util.Arrays;

int[][] myArray = {{1, 2}, {3, 4}, {5, 6}};
System.out.println(Arrays.deepToString(myArray));
```

Choose the method that best fits your needs. The `Arrays.toString()` and `Arrays.deepToString()` methods provide a convenient way to print array contents but may not be suitable for complex arrays or custom objects. In those cases, you might need to implement a custom `toString()` method or iterate over the array manually.