JAVA PROGRAMMING 1

Summer 2018 - Christian Hur

# Unit 8 Lecture - Arrays

Reading: Chapter 8

# What is an array?

* Array is a named list of data items (a collection of items), called elements, that have the same data type.
* Arrays are “**reference**” type like Objects.
* The array variable doesn’t store any value but references a memory address.
* Each item in the array is called an element.
* Data are stored in the elements of the array

# Array Declaration

**Syntax: Uninitialized arrays, no length specification**

*datatype*[] arrayVariable; //Java’s method

*datatype* arrayVariable[]; //C and C++ methods

**Syntax: Uninitialized arrays, length specification of 3 elements**

*datatype*[] arrayVariable = new *datatype*[3]; //3 elements

*datatype* arrayVariable[] = new *datatype*[3]; //3 elements

**Syntax: Initialized array with length of 3 elements - notice semi-colon**

*datatype*[] arrayVariable = {element1, element2, element3}; //3 elements

*datatype* arrayVariable[] = {element1, element2, element3}; //3 elements

# Arrays of primitive data types: int, float, double, boolean

\*Identifiers are usually “Plural”( + “type” if applicable).

int[] nums = new int[3]; //3 integer elements

float[] percents = new float[3]; //3 float elements

double[] scores = new double[3]; //3 double elements

boolean[] findings = new boolean[4]; //4 boolean elements

# Array Initialization

\*Initializing an array is also called populating the array.

int[] nums = {1,2,3}; //3 integer elements

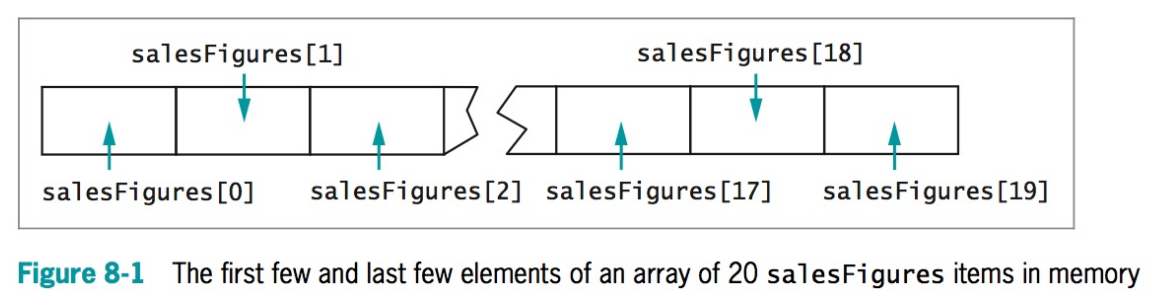
float[] percents = {1.5F,2.25F,5.35F}; //3 float elements

double[] scores = {98.55, 89.25,100.00}; //3 double elements

boolean[] findings = {true,false,false,true}; //4 boolean elements

# Accessing Array Elements

Each element in the array can be accessed using an array subscript (or index).



Array subscripts can be a numeric constant or integer variable. The subscript always starts from zero (0) to one less the lenght of the array.

double[] scores = {98.55, 89.25,100.00}; //3 double elements

System.out.print(scores[0]); //prints 98.55

System.out.print(scores[1]); //prints 89.25

System.out.print(scores[2]); //prints 100.00

int i = 0,j=1,k=2;

System.out.print(scores[i]); //prints 98.55

System.out.print(scores[j]); //prints 89.25

System.out.print(scores[k]); //prints 100.00

## Update an Array

scores[0] = 77.33;

scores[1] = 88.90;

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

## Length of Array

The length of an array can be determined using the “**length**” property.

double[] scores = {98.55, 89.25,100.00}; //3 double elements

scores.length; //3

Do not confuse this with the “**length()**” method for String.

double[] scores = {98.55, 89.25,100.00}; //3 double elements

int len = scores.lenght(); //won’t work, it only appliess to String

String name = “Christian”;

int len = name.length(); //9

## Cloning an Array

Be cautious when assigning an array to another array. Because arrays are a reference type, if an array is assigned to another array, they’re both referencing the same memory location and thus changing data in either array affects the other array content also.

Example:

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

int[] b = a; // b now points to a’s memory address

System.out.println(a[0]); // 1

System.out.println(b[0]); // 1

b[0] = 5; // also changes a[0]’s content

System.out.println(a[0]); // 5

System.out.println(b[0]); // 5

## Cloning an Array

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

int[] c = a.clone(); //c now gets a copy of a, not a’s reference

System.out.println(a[0]); // 1

System.out.println(c[0]); // 1

c[0] = 12; //does not affect a[0]

System.out.println(a[0]); // 1

System.out.println(c[0]); // 12

# Iterate Through the Array - For Loop, While Loop

You can use a **for loop** or **while loop** to iterate through the array to print out its data.

double[] scores = {98.55, 89.25,100.00}; //3 double elements

int len = scores.length;

for(int i=0; i < len; i++){

System.out.println(scores[i]);

}

int i = 0;

while(i < len){

System.out.println(scores[i]);

i++;

}

# Iterate Through the Array - Enhanced For Loop

You can use the enhanced FOR loop to iterate through the array to print out its data. This is often referred to as a **foreach loop**. This loop will print every element and so the length of the array is not specified.

**Syntax**:

for(datatype *val* : arrayName) {

//statements

}

* datatype - the matching datatype in the array
* val - a block variable that holds each element of the array.
* arrayName - this is the array to cycle through

**Example**:

double[] scores = {98.55, 89.25,100.00}; //3 double elements

for(int val: scores){

System.out.println(scores[i]);

}

# Arrays of objects: String, custom classes

String[] studentNames = new String[3];

String[] statesList = {“WI”,”CA”,”OR”,”WA”};

//Custom class

Employee[] emps = new Employee[3];

final int START\_NUM = 100;  
final double SALARY = 50000;

for(int i = 0; x < emps.length; ++i){

emps[i] = new Employee(START\_NUM + i,SALARY);

}

# Parallel Arrays

Arrays of different data types with related data. These data can be accessed using the same subscript.

**Example**: Arrays of student names, ids, grades, etc.

String[] names = new String[100];

String[] addresses = new String[100];

int[] id = new int[100];

char[] grades = new char[100];

A more efficient approach:

final int MAX\_INDEX = 100;

String[] names = new String[MAX\_INDEX];

String[] addresses = new String[MAX\_INDEX];

int[] id = new int[MAX\_INDEX];

char[] grades = new char[MAX\_INDEX];

# Searching an Array

You can iterate through the array and compare each element with another value to find a match.

int searchTerm = 304;

int[] points = {101, 108, 201, 213, 266, 304, 311, 409, 411, 412};

boolean found = false;

for(int i = 0; i < points.length; ++i) {

if(searchTerm == points[i])

found = true;

if(found)

break; //exit the loop prematurely

}

if(found)

System.out.print("Yes");

else

System.out.print("No");

# Passing Arrays to Methods

Array’s elements can be passed to methods

Example:

public void print(int n){

System.out.println(n);

}

int[] nums = {1,2,3};

print(nums[0]); //1 is passed to print

Primitive data types are always passed by value - never by reference in Java. You cannot pass primitive data types by reference (you can do this in some languages such as C, C++, C#, VB).

Passing the entire array to print() method:

print(nums);

The reference of the array is passed to the print() method, and so if any elements of the array is altered in the print() method, it’s also altered in the original array.

public void print(int[] n){

n[0] = 11; //The first element is changed

}

# Returning an Array from a Method

public int[] getNewArray(){

int[] temp = {3,4,5,6};

return temp;

}