Topics

- Arrays in Java

Arrays Introduction

innovate achieve lead

- Array → Collection of Homogeneous Data Elements
- Arrays are objects and implements only static (fixed-length) arrays
- Java follows strict bound checking for referencing array elements. If an attempt is made to reference the elements outside the bounds then "ArrayIndexOutOfBoundsException" will be thrown at run time.
- Arrays can be 1-dimensional (1-D), 2-dimensional (2-D) or multidimensional.
- Maximum Dimensions an Array can have = 255
- Arrays are Objects in Java. So, every element at each index is automatically initialized to some default value depending upon the type of array. [byte, short, int and long → 0, float, double → 0.0, booelan → false, char → '' (whitespace) and any class type array → null]
- <<length>> is the attribute of each array which can be referenced by

  <array-reference-name> . <length>

1-dimensional Arrays

innovate achieve lead

Syntax : One-Dimensional Arrays :

type[ ] arrayname = new type[Size];

or

type arrayname[ ] = new type[Size];

Examples :

1. int[ ] marks = new int[10]; // Each element is initialized to 0
   marks is an int type array, marks.length = 10, LB index =0 , UB index = 9
2. float[ ] values = new float[20]; // Each element is initialized to 0.0
   values is an float type array, values.length = 20, LB index =0 , UB index =19
3. double cgpa[ ] = new double[5]; // Each element is initialized to 0.0
   cgpa is double type array, cgpa.length = 5, LB index =0 , UB index = 4
4. Box[ ] boxes = new Box[20]; // Each element is initialized to 'null' value
   boxes is a Box type array, boxes.length = 20, LB index =0 , UB index = 19
5. Point points[ ] = new Point[20]; // Each element is initialized to 'null' value
   points is a Point type array, points.length = 20, LB index =0 , UB index =19
6. int[] marks = {10, 8, 6, 34, 0, 34};
   marks is a int type array, marks.length = 6, LB index = 0, UB index = 5

2-dimensional Arrays

innovate achieve lead

Syntax : Two-Dimensional Arrays :

type[ ][ ] arrayname = new type[Row_Size][Col_Size];

or

type arrayname[ ][ ] = new type[Row_Size][Col_Size];

Row index varies from 0 to Row_Size – 1

Column index varies from 0 to Col_Size – 1

Examples :

1. int[ ][ ] data = new int[3][3]; // Each element is initialized to 0

data *is 2-D int array, capacity = 9, row index 0 to 2 col index 0 to 2*

2. float values[][] = new float[10][4]; // Each element is initialized to 0.0

values *is 2-D float array, capacity = 40, row index 0 to 9 col index 0 to 3*

3. int table[][] = {{ 0,0,0},{1,1,1}}; // initializes first row to 0 & second to 1

2-dimensional Arrays ….

- Java treats a 2-D array as a collection of 1-D arrays.
- In two-dimensional arrays, individual rows can have variable number of elements <Variable Size Array>
- Example

```
int[][] data = new int[4][];          // Variable-Size Array, column-size not mentioned
        data is a 2-D array having 4 rows (row-index varies from 0 to 3)
data[0] = new int[10];                // Row-0 has 10 elements, column-index varies from 0 to 9
data[1] = new int[5];                 // Row-1 has 5 elements, column-index varies from 0 to 4
data[2] = new int[6];                 // Row-2 has 6 elements, column-index varies from 0 to 5
data[3] = new int[10];                // Row-3 has 10 elements, column-index varies from 0 to 9
```

data[1][6] → Results in ArrayIndexOutofBounds Exception

data[2][6] → Results in ArrayIndexOutofBounds Exception

Referring Array Elements

- 1-D Arrays, Syntax → array-name[<index>], where <index> varies from 0 (LB) to array-name.length-1(UB)
- 2-D Arrays, Syntax → array-name[<row-index>][<col-index], where <row-index> varies from 0 to no-of-rows -1, and <col-index> varies from 0 to no-of-cols -1.
- Example

1. int[] data = new int[5] ; data[0] = 0; ….., data[4] = 0;
2. boolean[] flags = new boolean[10]; flag[0] = false, ….., flags[9] = false
3. String[] names = new String[5]; names[0]=null, ……, names[4] = null
4. double[] values = {10.5, 5.6, 7.5, 4.5};
   values[0] = 10.5, values[1] = 5.6, values[2] = 7.5, values[3] = 4.5
5. int [][] marks = {{10,40,20},{25,56,57},{10,89,94}}
   marks[0][0] = 10, marks[0][1] = 40, marks[0][2] = 20,
   marks[1][0] = 25, marks[1][1] = 56, marks[1][2] = 57,
   marks[2][0] = 10, marks[2][1] = 89, marks[2][2] = 94

Displaying 1-D Array Elements

- Two Methods
  - ❑ Method 1: Using for { .. } loop

    Example :

```
int data[] = {10,6,8,9,-4,5};
for(int i=0; i<data.length; i++)
        System.out.println(data[i]);
```

  - ❑ Method 2: Using for each loop

Variable is automatically considered as of 'int' type The same variable is to be used inside the loop

Syntax : for(<type-of-array> <variable> : <array-name>)

Example :

```
int data[] = {10,6,8,9,-4,5};
for(int i : data)
        System.out.println(i);
```

Displaying 1-D Array Elements : Example 1


```
// File Name : Demo.java
class Demo
{
        public  static  void  main(String[] args)
        {
                boolean[]  flags =  new  boolean[5];

                // Method-1 : Using for loop
                System.out.println("Output by Method 1");
                for(int i =0; i < flags.length; i++)
                        System.out.println(flags[i]);

                // Method-2 : Using for each loop
                System.out.println("Output by Method 2");
                for(boolean k : flags)
                        System.out.println(k);
        }// End of Method
}// End of class Demo
```

<<OUTPUT>>

```
Output by Method 1
false
false
false
false
false
Output by Method 2
false
false
false
false
false
```

Displaying 1-D Array Elements : Example 2


```
// File Name : Demo.java
class Demo
{
        public    static    void      main(String[] args)
        {
                String[]  names =  new String[5];

                // Method-1 : Using for loop
                System.out.println("Output by Method 1");
                for(int i =0; i < names.length; i++)
                        System.out.println(names[i]);

                // Method-2 : Using for each loop
                System.out.println("Output by Method 2");
                for(String k : names)
                        System.out.println(k);
        }// End of Method
}// End of class Demo
```

```
F:\>java Demo
Output by Method 1
null
null
null
null
null
Output by Method 2
null
null
null
null
null
```

Displaying 2-D Array Elements

- Two Methods
  - ❑ Method 1: Using nested for { .. } loop

    Example :

```
int data[][] = new int[5][5];
System.out.println(data.length);                    // Displays 5 not 25
for(int i=0; i<data.length; i++)                    // data.length = no of rows
        for(int j=0; j<data[i].length; j++)         // each jth column under ith row
                System.out.println(data[i][j]);
```

  - ❑ Method 2: Using nested for each loop

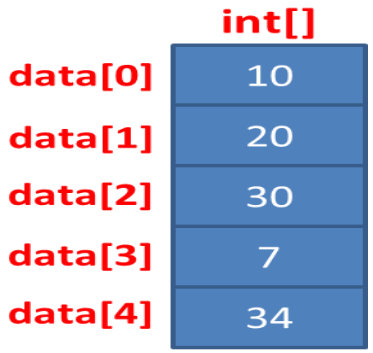
```
Syntax :    for(<type-of-array> [ ]  <variable-i> : <array-name>)
                    for(<type-of-array> <variable-j> : <variable-i>
```

Displaying 2-D Array Elements : Example

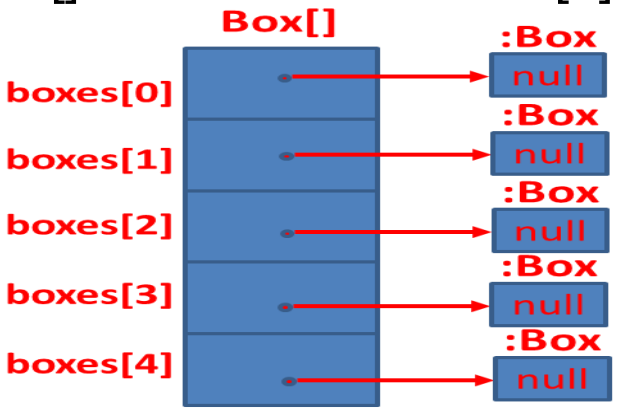

```
// File Name : Demo.java
class Demo
{
        public      static      void        main(String[] args)
        {
                int[][] data = new int[5][5];

                System.out.println(data.length);      // Displays 5

                // Method 1
                System.out.println("Method 1 : Using Nested for Loop");
                for(int i = 0; i < data.length; i++)
                        for(int j = 0; j < data[i].length; j++)
                                System.out.println(data[i][j]);

                // Method 2
                System.out.println("Method 2 : Using Nested for-each Loop");
                for(int[] i : data)
                        for(int j : i)
                                System.out.println(j);
        }// End of Method
}// End of class Demo
```

Primitive Type vs Object Type Array

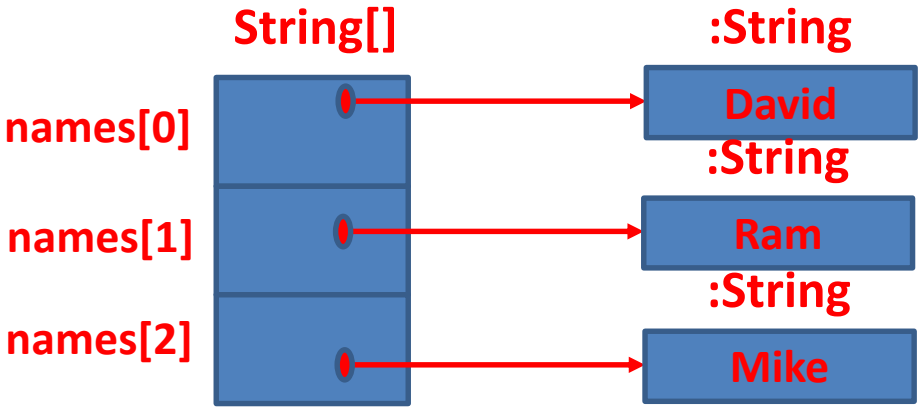
- int[] data = { 10, 20, 30, 7, 34 };



- Box[] boxes = new Box[5];



- String[] names = { “David”, “Ram”, “Mike”};



Thank You