Array

Department of Data Science
& Engineering, DCA, MIT

Arrays

- A group of variables containing values that all have the same type
- Arrays are fixed-length entities
- In Java, arrays are objects, so they are considered reference types
- But the elements of an array can be either primitive types or reference types

Arrays

- We access the element of an array using the following syntax
  - name[index]
  - “index” must be a nonnegative integer
    - “index” can be int/byte/short/char but not long
- In Java, every array knows its own length
- The length information is maintained in a public final int member variable called **length**

Declaring and Creating Arrays

- int c[ ] = new int [12]
  - Here, “c” is a reference to an integer array
  - “c” is now pointing to an array object holding 12 integers
  - Like other objects arrays are created using “new” and are created in the heap
  - “int c[ ]” represents both the data type and the variable name. Placing number here is a syntax error
  - **int c[12]; // compiler error**

Declaring and Creating Arrays

- int[ ] c = new int [12]
  - Here, the data type is more evident i.e. “int[ ]”
  - But does the same work as
    - int c[ ] = new int [12]
- Is there any difference between the above two approaches?

Declaring and Creating Arrays

- int c[ ], x
  - Here, ‘c’ is a reference to an integer array
  - ‘x’ is just a normal integer variable
- int[ ] c, x;
  - Here, ‘c’ is a reference to an integer array (same as before)
  - But, now ‘x’ is also a reference to an integer array

Arrays

ArrayDemo.java

```
1  public class ArrayDemo {
2      public static void main(String[] args) {
3          int [] a = new int[10];
4          for (int i = 0; i < a.length; i++) {
5              a[i] = i;
6          }
7          for (int i = 0; i < a.length; i++) {
8              System.out.println(a[i]);
9          }
10     }
11 }
12
```

Using an Array Initializer

- We can also use an array initializer to create an array
  - int n[ ] = {10, 20, 30, 40, 50}
- The length of the above array is 5
- n[0] is initialized to 10, n[1] is initialized to 20, and so on
- The compiler automatically performs a “new” operation taking the count information from the list and initializes the elements properly

Arrays of Primitive Types

- When created by “new”, all the elements are initialized with default values
  - byte, short, char, int, long, float and double are initialized to zero
  - boolean is initialized to false
- This happens for both member arrays and local arrays

Arrays of Reference Types

- String [] str = new String[3]
  - Only 3 String references are created
  - Those references are initialized to **null** by default
  - Need to explicitly create and assign actual String objects in the above three positions.
    - str[0] = new String(“Hello”);
    - str[1] = “World”;
    - str[2] = “I” + “ Like” + “ Java”;

Passing Arrays to Methods

void modifyArray(double d[ ]) {…}

double [] temperature = new double[24];

modifyArray(temperature);

- Changes made to the elements of ‘d’ inside “modifyArray” is visible and reflected in the “temperature” array
- But inside “modifyArray” if we create a new array and assign it to ‘d’ then ‘d’ will point to the newly created array and changing its elements will have no effect on “temperature”

Passing Arrays to Methods

- Changing the elements is visible, but changing the array reference itself is not visible

```
void modifyArray(double d[ ]) {
    d[0] = 1.1; // visible to the caller
}
void modifyArray(double d[ ]) {
    d = new double [10];
    d[0] = 1.1; // not visible to the caller
}
```

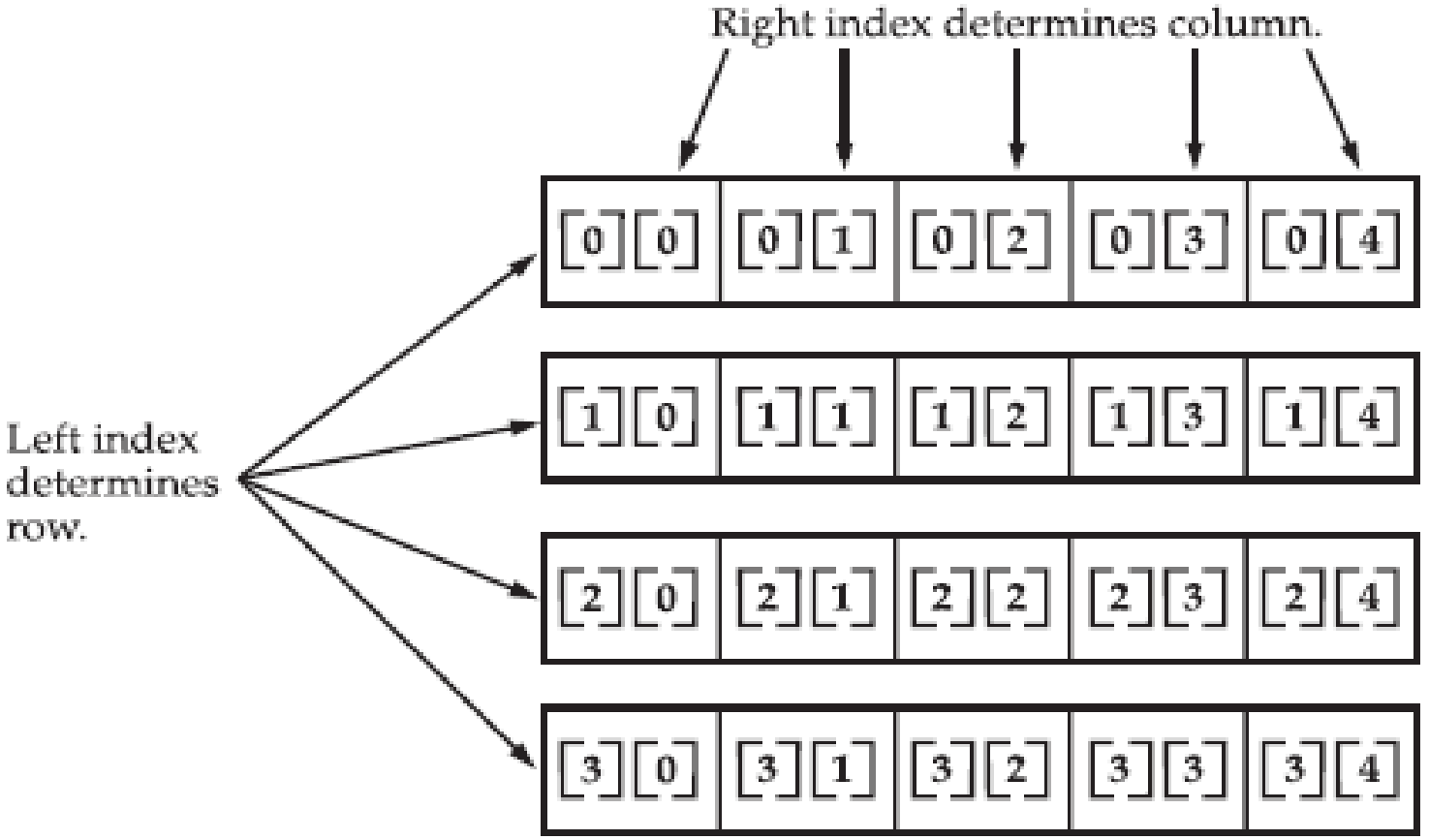
Multidimensional Arrays

- Can be termed as array of arrays.
- int b[ ][ ] = new int[3][4];
  - Length of first dimension = 3(rows)
    - b.length equals 3
  - Length of second dimension = 4 (columns)
    - b[1].length equals 4
- int[ ][ ] b = new int[3][4];
  - Here, the data type is more evident i.e. “int[ ][ ]”

Multidimensional Arrays

- int b[ ][ ] = { { 1, 2, 3 }, { 4, 5, 6 } };
  - b.length equals 2
  - b[0].length and b[1].length equals 3
- All these examples represent rectangular two dimensional arrays where every row has same number of columns
- Java also supports jagged array where rows can have different number of columns

Multidimensional Arrays



Given: int twoD [ ] [ ] = new int [4] [5] ;

Multidimensional Arrays

Example – 1

```
int b[ ][ ];
b = new int[2][ ];
b[0] = new int[2];
b[1] = new int[3];
b[0][2] = 7; //will throw an exception
```

Example – 2

```
int b[ ][ ] = { { 1, 2 }, { 3, 4, 5 } };
b[0][2] = 8; //will throw an exception
```

In both cases

b.length equals 2
b[0].length equals 2
b[1].length equals 3