Arrays in Java

Prof. Kaustubh Kulkarni

Array

- An array is a group of like-typed variables that are referred to by a common name.
- Arrays of any type can be created and may have one or more dimensions.
- A specific element in an array is accessed by its index.
- Arrays offer a convenient means of grouping related information.
  - E.g. Student scores

One-Dimensional Arrays

- A one-dimensional array is a list of like-typed variables.
- To create an array, you first must create an array variable of the desired type.
- The general form of a one-dimensional array **declaration** is

  type[ ] *var-name*;

- Here, *type* declares the element type (also called the base type) of the array.
- For example, the following **declares** an array named month_days with the type “array of int”:

  int[] month_days;

Allocation

- Although the **declaration** on the previous slide establishes the fact that **month_days** is an array variable, no physical array actually exists.
- **new** is a special operator that allocates memory.
- The general form of **new** as it applies to one-dimensional arrays is as follows:

  array-var = new type [size];
- This example allocates a 12-element array of integers and links them to month_days:

  month_days = new int[12];

It is possible to combine the **declaration** of the array variable with the **allocation** of the array itself:

int[] month_days = new int[12];

Accessing elements

- The elements in the array allocated by **new** will automatically be initialized to zero (for numeric types).
- Once you have allocated an array, you can access a specific element in the array by specifying its index within square brackets.
- Array indices start at zero.
- For example, this statement assigns the value 28 to the second element of month_days:

```
month_days[1] = 28;
```

- The next line displays the value stored at index 3:

```
System.out.println(month_days[3]);
```

Array Initializer

- Arrays can be initialized when they are declared.
- An array initializer is a list of comma-separated expressions surrounded by curly braces.
- The commas separate the values of the array elements.
- The array will automatically be created large enough to hold the number of elements you specify in the array initializer.
- There is no need to use **new**.

Example of array initializer

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

Another Example

```
// Average an array of values.
class Average {
 public static void main(String[] args) {
 double[] nums = {10.1, 11.2, 12.3, 13.4, 14.5};
 double result = 0;
 int i;
for(i=0; i<5; i++)
 result = result + nums[i];
 System.out.println("Average is " + result / 5);
 }
}
```

Array Index Out of Bounds

- Java strictly checks to make sure you do not accidentally try to store or reference values outside of the range of the array.
- The Java run-time system will check that all array indexes are in the correct range.
- For example, the run-time system will check the value of each index into month_days to make sure that it is between 0 and 11 inclusive.
- If you try to access elements outside the range of the array (negative numbers or numbers greater than the length of the array), you will cause a run-time error.

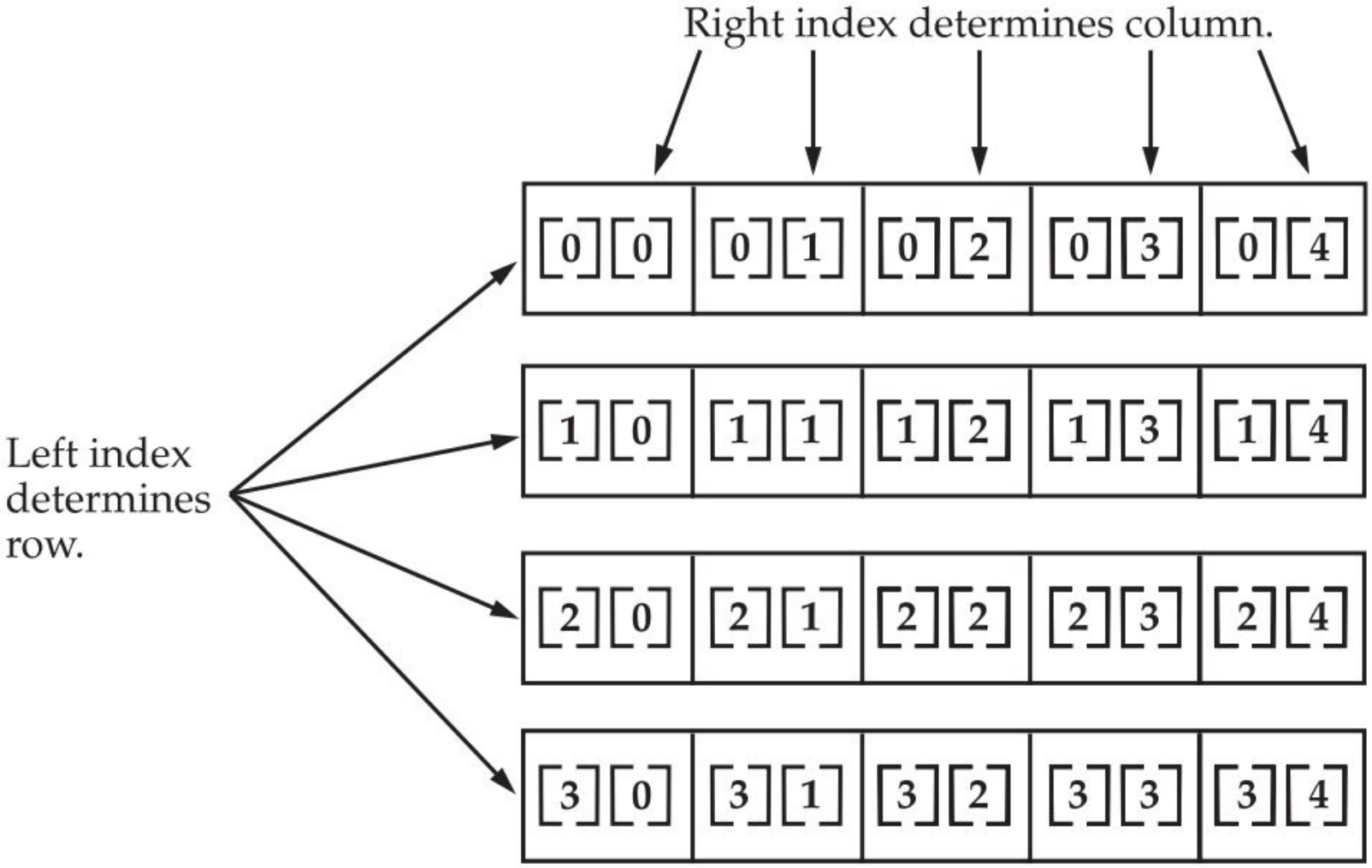
Multidimensional Arrays

- In Java, multidimensional arrays are implemented as arrays of arrays.
- To declare a multidimensional array variable, specify each additional index using another set of square brackets.
- For example, the following declares a two-dimensional array variable called twoD:

```
int[][] twoD = new int[4][5];
```

- This allocates a 4 by 5 matrix and assigns it to twoD.
- Internally, this matrix is implemented as an array of arrays of int.
- Conceptually, this array will look like the one shown in the next slide.

A conceptual view of a 2D array



2D array example

```
class TwoDArray {
    public static void main(String[] args) {
    int[][] twoD= new int[4][5];
    int i, j, k = 0;
    for(i=0; i<4; i++)
    for(j=0; j<5; j++) {
    twoD[i][j] = k;
    k++;
    }
    for(i=0; i<4; i++) {
    for(j=0; j<5; j++)
    System.out.print(twoD[i][j] + " ");
    System.out.println();
    }
    }
   }
```

- int[][] twoD= new int[4][5];

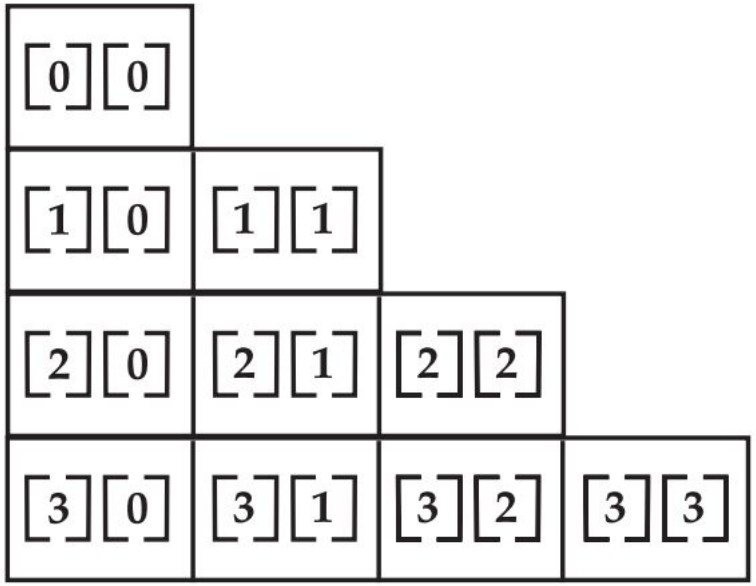
  is equivalent to:
- int[][] twoD = new int[4][];
- twoD[0] = new int[5];
- twoD[1] = new int[5];
- twoD[2] = new int[5];
- twoD[3] = new int[5];

Irregular Array

- When you allocate memory for a multidimensional array, you need only specify the memory for the first (leftmost) dimension.
- You can allocate the remaining dimensions separately.
- As stated earlier, since multidimensional arrays are actually arrays of arrays, the length of each array is under your control.
- Each “row” (1st dimension) can have different number of “columns” (2nd dimension)
- AKA Jagged Array

Example of a Jagged Array

- int[][] twoD = new int[4][];
- twoD[0] = new int[1];
- twoD[1] = new int[2];
- twoD[2] = new int[3];
- twoD[3] = new int[4];



```
// Manually allocate differing size second dimensions.
class TwoDAgain {
 public static void main(String[] args) {
 int[][] twoD = new int[4][];
 twoD[0] = new int[1];
 twoD[1] = new int[2];
 twoD[2] = new int[3];
 twoD[3] = new int[4];
 int i, j, k = 0;
for(i=0; i<4; i++)
 for(j=0; j<i+1; j++) {
 twoD[i][j] = k;
 k++;
 }
 for(i=0; i<4; i++) {
 for(j=0; j<i+1; j++)
 System.out.print(twoD[i][j] + " ");
 System.out.println();
 }
 }
}
```

When to use

- Irregular arrays can be used effectively in some situations.
- For example, if you need a very large two-dimensional array that is sparsely populated (that is, one in which not all of the elements will be used), then an irregular array might be a perfect solution.

Initializing multidimensional arrays

- Simply enclose each dimension's initializer within its own set of curly braces.
- double[][] m = {
- { 0*0, 1*0, 2*0, 3*0 },
- { 0*1, 1*1, 2*1, 3*1 },
- { 0*2, 1*2, 2*2, 3*2 },
- { 0*3, 1*3, 2*3, 3*3 }
- };
- It creates a matrix where each element contains the product of the row and column indexes.
- Also notice that you can use expressions as well as literal values inside of array initializers.

Alternative Array Declaration Syntax

- There is a second form that may be used to declare an array:

  type var-name[ ];

- Here, the square brackets follow the array variable name, and not the type specifier.
- For example, the following two declarations are equivalent:

```
int a1[] = new int[3];
int[] a2 = new int[3];
```

- The following declarations are also equivalent:

```
char twod1[][] = new char[3][4];
char[][] twod2 = new char[3][4];
```

- This alternative declaration form offers convenience when converting code from C/C++ to Java.
- It also lets you declare both array and non-array variables in a single declaration statement.

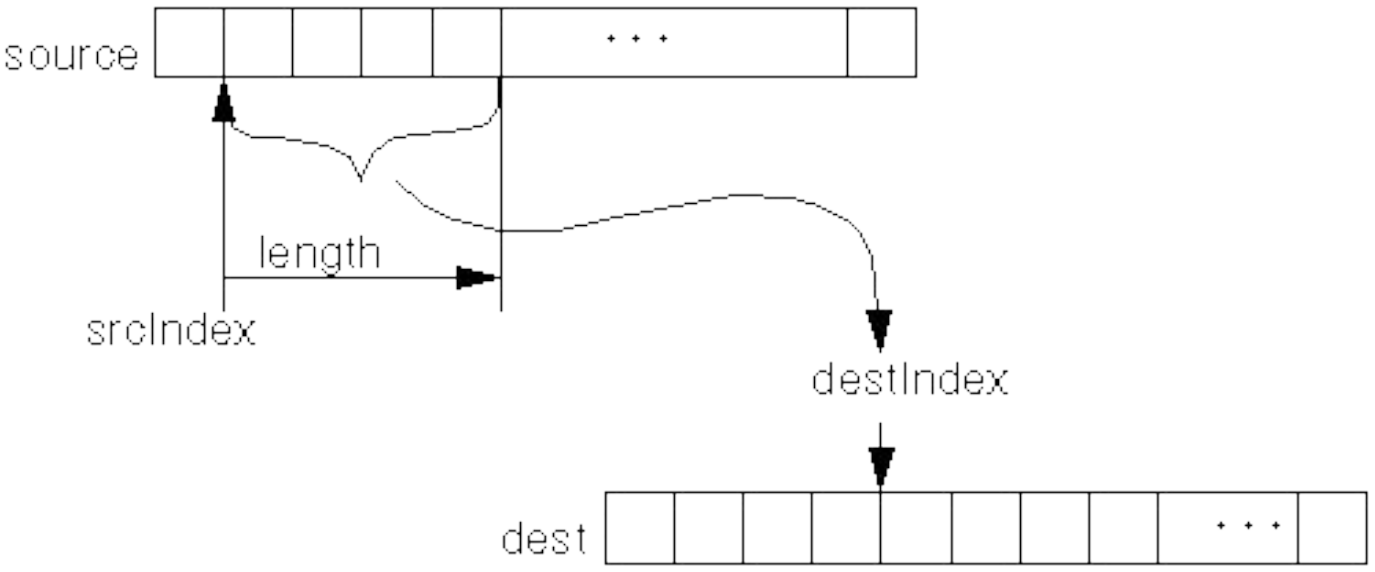
Using arraycopy( )

- The arraycopy( ) method can be used to copy quickly an array of any type from one place to another.
- This is much faster than the equivalent loop written out longhand in Java.
- Here is an example of two arrays being copied by the arraycopy( ) method.
- First, a is copied to b.
- Next, all of a’s elements are shifted down by one.
- Then, b is shifted up by one.

Using arraycopy( )

- Use System's arraycopy() method to efficiently copy data from one array into another.
- The arraycopy() method requires five arguments:
- public static
- void arraycopy(Object source,
- int srcIndex,
- Object dest,
- int destIndex,
- int length,
- The two Object arguments indicate the array to copy from and the array to copy to.
- The three integer arguments indicate the starting location in each the source and the destination array, and the number of elements to copy.

Illustration of arraycopy()



```
class ArrayCopyTest {
    public static void main(String[] args) {
        byte[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a', 't', 'e', 'd' };
        byte[] copyTo = new byte[7];

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo, 0));
    }
}
```