UBC OKANAGAN

COSC 111
Computer Programming I

Arrays

Dr. Firas Moosvi

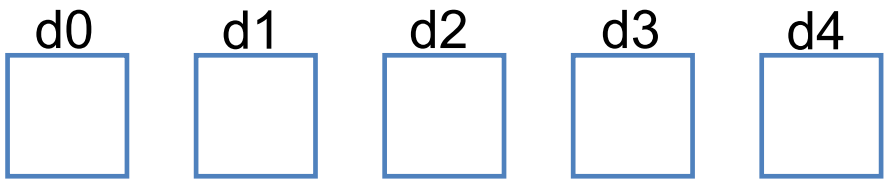
Acknowledgement: Slides (provided by Dr. Abdallah Mohammed) mainly rely on the materials prepared by Y. D. Liang for the textbook "Introduction to Java Programming, 10th Ed., Pearson Edu. Inc.".

Arrays Overview

Suppose you need many variables in your program.

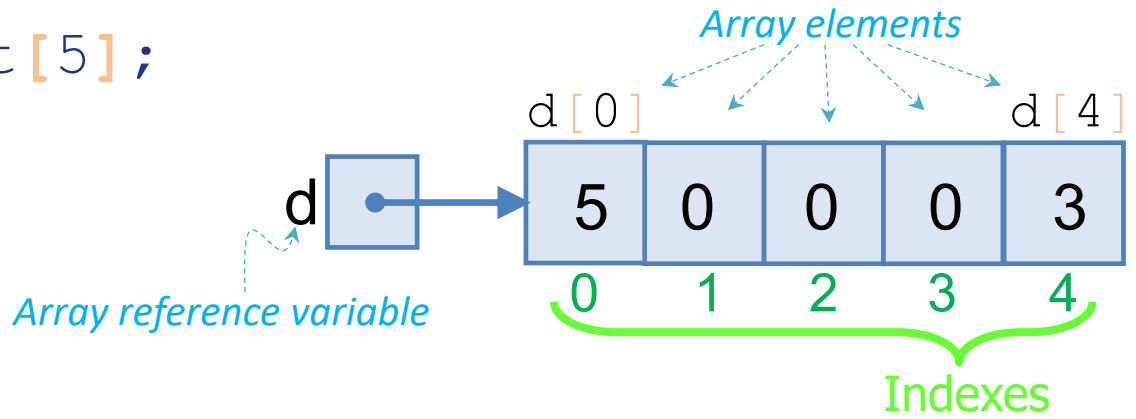
You could either create a separate name for each variable:

- `int d0, d1, d2, d3, d4;`



Or you could create an array that has multiple elements

- `int[] d = new int[5];`
- `d[0] = 5;`
- `d[4]= 3;`



Declaring, Creating, and Initializing Arrays

Creating an Array

Array is a **data structure** that represents a collection of the **SAME types of data**.

Creating an array involves 2 steps:

(1) Declare a **variable to *reference*** the array:

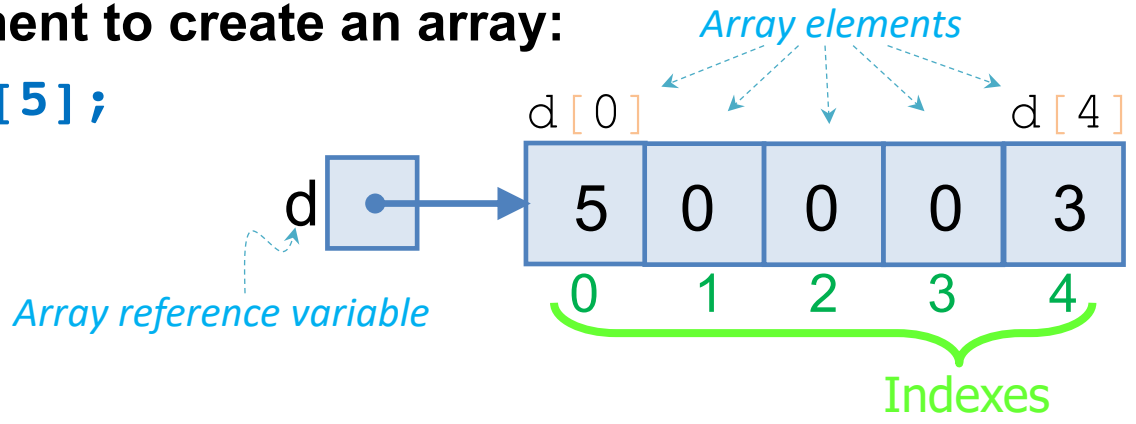
```
int[] d;                    //just the pointer – no array object yet.
```

(2) Create the **array object**

```
d = new int[5];  //Now d is referring to a 5 element array object
```

You can also use one statement to create an array:

```
int[] d = new int[5];
```



How to access array elements? use array name followed by the index.

```
d[1] = d[0] + d[4];  //now d[1] is 8
```

Initializing an Array

Default values: When an array is created, its elements are assigned the default value of

- **0** for the numeric primitive data types,
- **\u0000** for **char** types, and
- **false** for **boolean** types.

You can declare, create, and initialize an array in one step:

```
int[] d = {5, 2, 10, 7, 12};
```

This shorthand notation is equivalent to the following statements:

```
int[] d = new int[5];
ar[0] = 5;
…
ar[4] = 12;
```

Caution: using the shorthand notation, you have to declare, create, and initialize the array **all in one statement**.

```
double[] d;
d = {2, 5, 3.4, 3.5}; //this is wrong!
```

Array Length and Indices

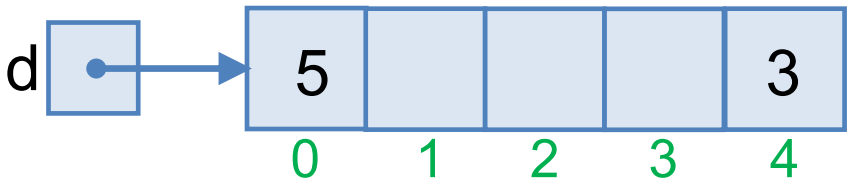
Once an array is created, **its size is fixed**.

You can find the size using, for example:

```
d.length
```

The array indices are 0-based, i.e., it starts from 0 to *length-1*.

- The array d, for example, has 5 int values, and the indices are from 0 to 4.
- This means the index of the last element of the array is equal to the array length – 1



Practice

1) Create an `int` array with name `ages` with 20 elements.

2) Set the value of the 1st element to 12.

3) Set the value of the last element to 1.

4) Set the 2nd element to a value obtained from the user

5) How do you know how many elements are in an array?

Clicker Question

What is the size of this array?

```
int[] myArray = new int[10];
```

A. 9

B. 10

C. 11

D. error

Clicker Question

What are the contents of this array?

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

A. error

B. 0, 1, 2, 3

C. 1, 2, 3, 4

D. 4, 3, 2, 1

Processing Arrays

Processing Arrays

1) Using a for loop: it is preferred to use a for loop to process array elements when several elements are processed in the **same fashion repeatedly**.

```
for (int i=0; i<arrayRefVar.length; i++){
   //same actions applied to all elements evenly
}
```

- **For example:** this code Initializes an array *ar* with random values :

```
for (int i=0; i<ar.length; i++)
   ar[i] = Math.random() * 100;
```

2) Without a for loop: if you are applying **different actions** to the array elements, you should **probably avoid using the loop**.

Processing Arrays: Examples

Initializing an array with random values:

```
for (int i=0; i<ar.length; i++)
    ar[i] = Math.random() * 100;
```

Initializing arrays with input values

```
Scanner input = new Scanner(System.in);
System.out.print("Enter "+ar.length+" values: ");
for (int i=0; i<ar.length; i++)
    ar[i] = input.nextDouble();
```

Printing arrays

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

Processing Arrays: Examples, cont.

Summing all elements

```
int total = 0;
for(int i=0; i<ar.length; i++)
    total += ar[i];
```

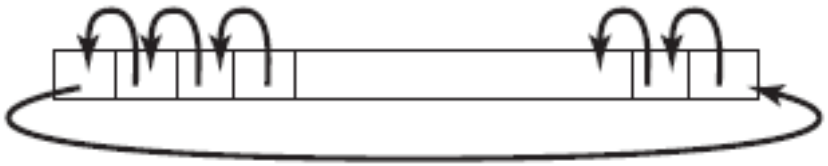
Finding the largest element

```
int max = ar[0];
for (int i=1; i<ar.length; i++)
    if (ar[i] > max)
        max = ar[i];
```

Processing Arrays, cont.

Shifting the elements one position to the left and filling the last element with the first element:

```
// Retain the first element
int temp = ar[0];
// Shift elements left
for(int i=1; i<ar.length; i++)
   ar[i - 1] = ar[i];
// Move the first element to fill in the last position
ar[ ar.length-1 ] = temp;
```



Practice

Write a program that reads *n* elements (real numbers) and then displays their average **along with the number of elements above average**. Your program should begin by asking the use to enter the value *n.*

```
Enter the number of elements: 4
Enter element 1:5.2
Enter element 2:1.5
Enter element 3:4.4
Enter element 4:2.7
The average is: 3.45
Number of items above the average is 2
```

Practice, cont.

Basic idea:

If only the average is required, then we will run a simple for loop similar to what we did before, and we don't need arrays. However, to get the number of items above the average, we need to remember these numbers at the end and compare them with the average. To remember the entries, store them in an array.

The algorithm:

- 1. Prompt the user for the number of items
- 2. Read the values from the user:
  - Store them in an array.
  - Find the sum
- 3. Find the average.
- 4. Count the number of elements above average.
- 5. Display the output.

Practice, cont.

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    // 1. prompt the user for the number of items
    System.out.print("Enter the number of elements: ");
    int n = input.nextInt();
    double[] numbers = new double[n];
    double sum = 0;
    // 2. Read the values from the user
    for (int i = 0; i < numbers.length; i++) {
        System.out.print("Enter element " + (i+1) + ":");
        numbers[i] = input.nextDouble();
        sum += numbers[i];
    }
    // 3. Find the average
    double average = sum / n;
    // 4. Count the number of elements above average
    int count = 0;
    for (int i = 0; i < numbers.length; i++)
        if (numbers[i] > average)
            count++;
    // 5. Display the output
    System.out.println("The average is: " + average);
    System.out.println("Number of items above the average is " + count);
    input.close();
}
```

Clicker Question

What are the contents of this array?

```
int[] nums = new int[5];
for(int i = 0; i <= 5; i++)
    nums[i] = 5-i;
```

A. error

B. 0, 1, 2, 3, 4

C. 0, 1, 2, 3, 4, 5

D. 4, 3, 2, 1, 0

E. 5, 4, 3, 2, 1, 0

Clicker Question

What are the contents of this array?

```
int[] nums = new int[6];
for(int i = 2; i <= 4; i++)
    nums[i] = i-1;
```

A. error

B. 0, 1, 2, 3, 0, 0

C. 0, 0, 1, 2, 3, 0

D. 0, 0, 0, 1, 2, 3

E. 0, 1, 2, 3, 4, 5

For-Each Loops

For-each Loops

for-each loops enable you to traverse the complete array sequentially **without using an index variable**.

```
for (elementType value: arrayRefVar) {
   // Process the value
}
```

Example: display all elements in ar:

```
for (int item: ar)
   System.out.println(item);
```

Note that **`value`** is only a copy of the array-element. Therefore, if **`value`** is a primitive and we change it, the original array-element is not changed.

Clicker Question

What are the contents of the two arrays x and y?

```
int[] x = {1,1,1}, y = {1,1,1};
for(int item: x)
    item++;
for(int i = 0; i < y.length; i++)
   y[i]++;
```

A. x: 1,1,1 y: 1,1,1

B. x: 2,2,2 y: 2,2,2

C. x: 1,1,1 y: 2,2,2

D. x: 1,2,3 y: 2,2,2

Arrays are Objects!!

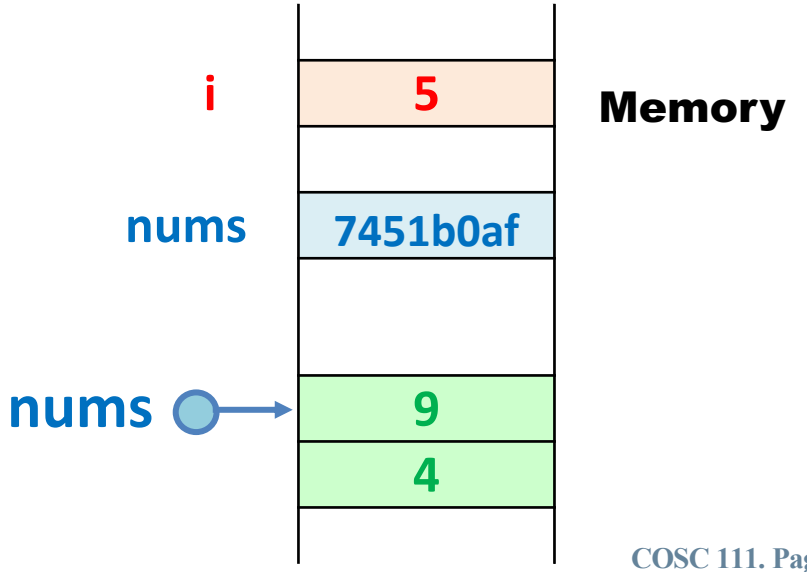
Arrays are Objects, not primitive types!

Java's types are divided into:

- 1. Primitive types
  - Includes `boolean`,`byte`,`char`,`short`,`int`,`long`,`float` and `double`.
  - A primitive-type variable stores **a value** of its declared type.
- 2. Reference types
  - Includes all non-primitive types, (e.g., **Arrays**, Strings, Scanner, etc.)
  - A reference-type variable stores data which Java uses to find the object in the memory.
  - Such a variable is said to **refer to an object** in the program.

- Exampe:

```
int i = 5;
int[] nums = { 9 , 4 };
```



Declaring, Creating, and Working with Arrays

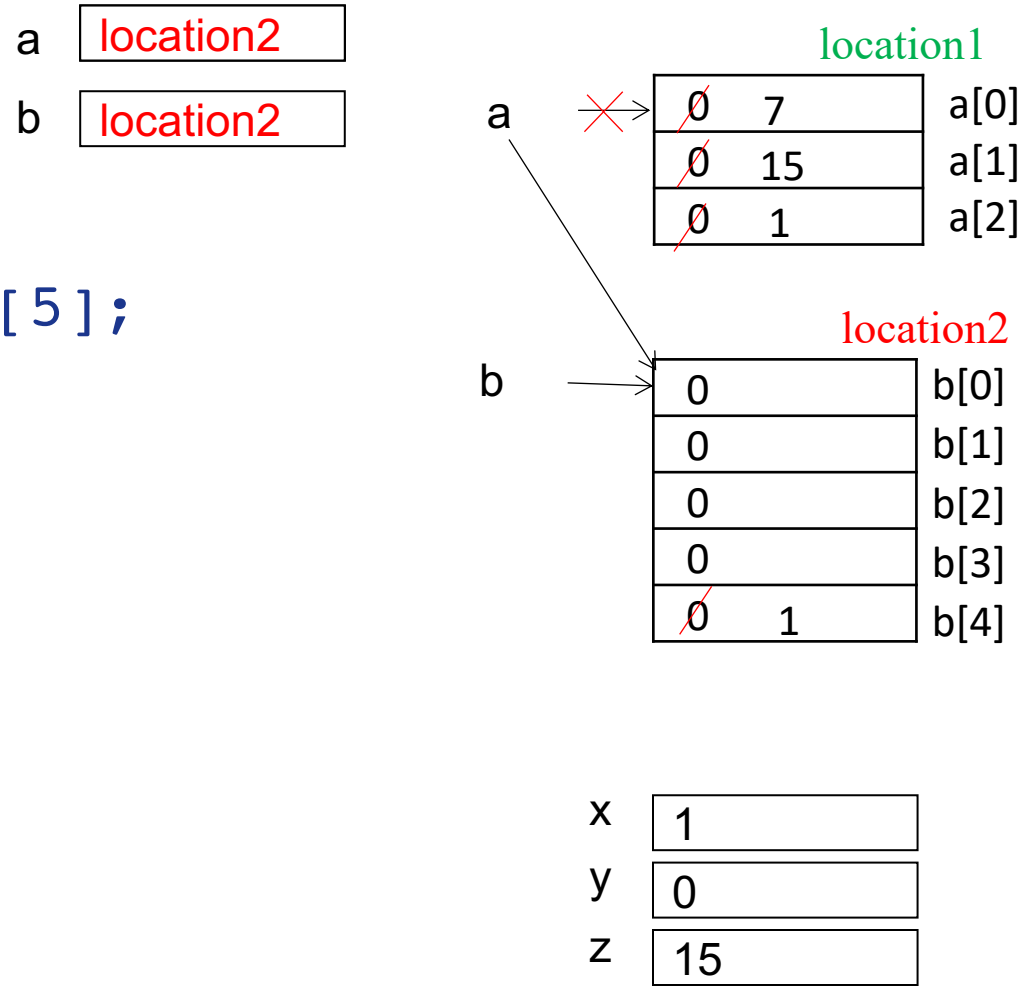
```
int[] a ;
a = new int[3];

int[] b = new int[5];

a[0] = 7;
a[1] = a[0] + 8;
a[2]++;

int x = a[2];
int y = a[2] - 1;
int z = a[2-1];

a = b;
a[a.length-1]++;
```



Clicker Question

How many array references and objects do we have at the end of this code?

A. 3 references, 3 objects

B. 3 references, 2 objects

C. 3 references, 1 object

D. 2 references, 2 objects

E. Error

```
int[] a = new int[6];
int[] b = new int[4];
int[] c = new int[6];
a = b;
a = c;
for(int i=0; i<4; i++)
   b[i] = a[i];
```

Clicker Question

How many array references and objects do we have at the end of this code?

A. 3 references, 3 objects

B. 3 references, 2 objects

C. 3 references, 1 object

D. 2 references, 2 objects

E. Error

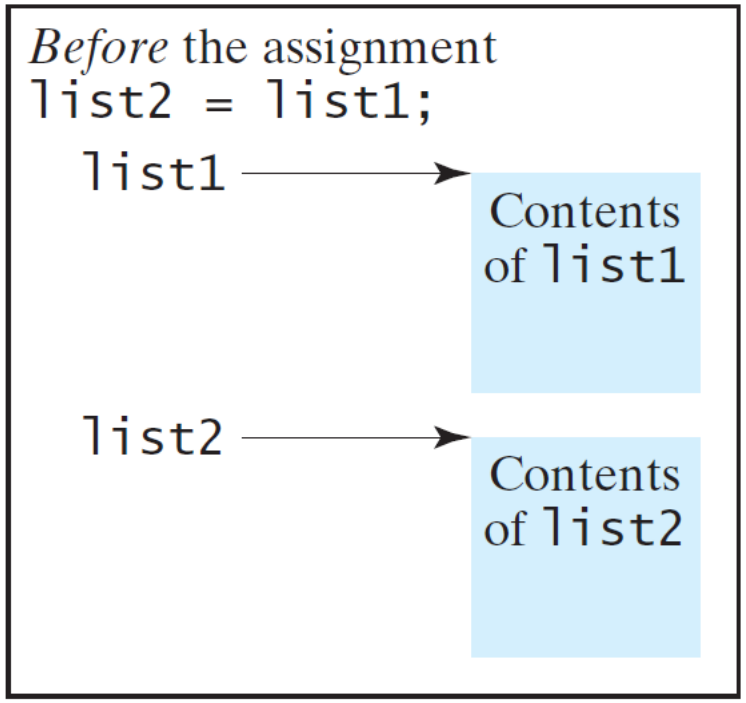
```
int[] a = new int[6];
int[] b = {1,2,3,4,5};
int[] c = b;
for(int i=0; i<4; i++)
   c[i] = a[i];
a = b;
```

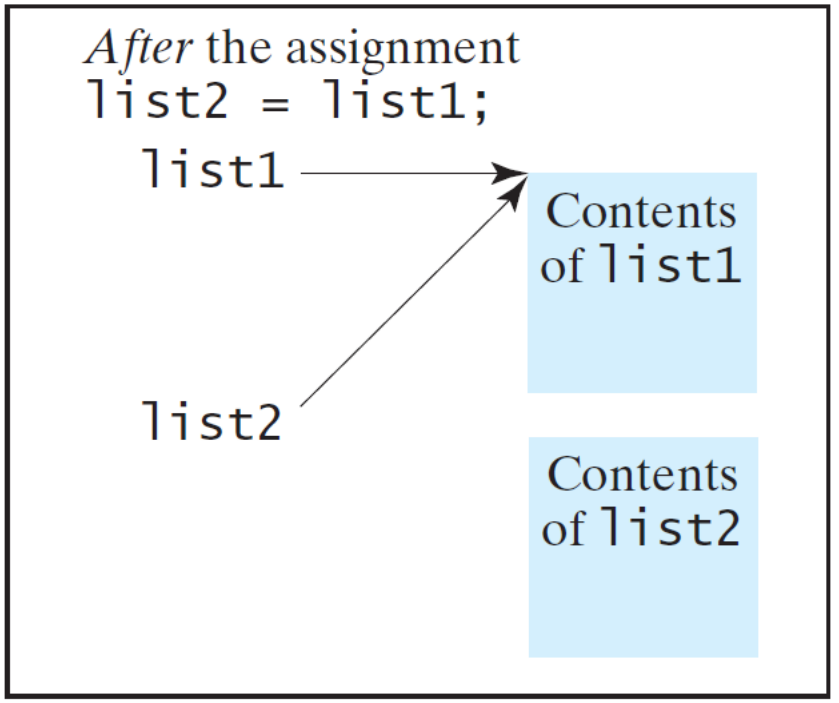
Copying Arrays

Copying Arrays

Given two arrays list1 and list 2, the statement, **`list2 = list1;` does not copy the contents** of **list1** to **list2**, but instead merely copies the reference value from **list1** to **list2**

- *garbage collection:* An array that has no reference becomes garbage, which will be collected by the Java Virtual Machine.





Copying Arrays, cont.

You may use of the following ways to copy arrays:

1. Use a loop to **copy individual elements** one by one.
2. Use the **`arraycopy`** method in the **System** class.

   Syntax of arraycopy is as follows:

   arraycopy(sourceArray, srcPos, targetArray, tarPos, length);

Example: consider the following two arrays:

```
int[] source = {2, 3, 1, 5, 10};
int[] target = new int[source.length];
```

(1) copying using a loop:

```
for (int i = 0; i < source.length; i++)
  target[i] = source[i];
```

(2) copying using arraycopy:

```
System.arraycopy(source, 0, target, 0, source.length);
```

Practice

```
public class Ex3_createIdenticalArray {
    public static void main(String[] args) {
        int[] x = { 1, 2, 3, 4 };
        // write code to create an identical array y
    }
}
```

Practice

Assume you have an integer array a that includes any number of elements, e.g. a = {1,2,3,4,5}.

1) Write a program that creates a new array b that has the same elements in a but in reverse order; e.g. b = {5,4,3,2,1}. Your code should be general enough to work with any values of a.

2) Write a program that creates a new array c that has the same elements of a but separated by 0's, e.g. c = {1,0,2,0,3,0,4,0,5,0}

Arrays and Methods

Passing Arrays to Methods

Java uses **pass-by-value** to pass arguments to a method.

If the parameter is of:

- a primitive type:
  - the actual value is passed. Changing the value of the local parameter inside the method does not affect the value of the variable outside the method.
- an array type,
  - the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body will affect the original array that was passed as the argument.
  - **The array in the method is the SAME AS the array being passed**.

Passing Arrays to Methods, cont.

In the following example, x will not change within the method, but y will change. Can you explain why?

```
public class PassArraysToMethods {
    public static void main(String[] args) {
        int x = 1; // x is a primitive variable
        int[] y = new int[10]; // y is an array
        m(x, y); // pass x and y to method m
        System.out.println("x is " + x);
        System.out.println("y[0] is " + y[0]);
    }

    public static void m(int a, int[] b) {
        a = 4444; // Assign a new value to a
        b[0] = 5555; // Assign a new value to b[0]
    }
}
```

Output:

```
x is 1
y[0] is 5555
```

Practice

Write two methods that return the minimum and maximum elements of an array with the following headers:

```
public static int min(int[] ar)
public static int max(int[] ar)
```

Test your methods with appropriate data.

Basic idea for min:

- Declare a variable *minElem* and assign the first array element to it.
- Compare *minElem* to the remaining elements of the array. Whenever an element has less value, store that value in *min*.

Basic idea for max:

Similar to that of min except that we find the maximum element.

Practice, cont.

```
public class minmax {
    public static void main(String[] args) {
        int[] x = {1, 4, 2, 7, -1, 0};
        System.out.println("Min element:" + min(x));
        System.out.println("Max element:" + max(x));

    }
    public static int min(int[] ar){
        int minElem = ar[0];
        for (int i = 1; i < ar.length; i++) {
            if(minElem > ar[i])
                minElem = ar[i];
        }
        return minElem;
    }
    public static int max(int[] ar){
        int minElem = ar[0];
        for (int i = 1; i < ar.length; i++) {
            if(minElem < ar[i])
                minElem = ar[i];
        }
        return minElem;
    }
}
```

Practice, cont.

1) What happens if we change the values of ar[i] within any of the two methods?

2) What should be changed in the methods if we want to find the min and max of the array of double values {1.3, 1.8, 2.4, 4.2} ?

3) How to have two methods for min (and two for max), one to handle integers and the other for doubles?

Returning an Array from a Method

Based on the discussion in this section, when a method returns an array **the reference of the array is returned**.

Sample method that returns an array:

```
Public void int[] amethod(){
    int[] x = {1, 2, 4};
    return x;
}
```

Practice

Write a method that has the following header:

public static int[] getRandomArray(int n, int from, int to)

The method should return an array of *n* random numbers from *from* to *to*.

Test your program by calling the method and printing out its elements.

Remember:

```
a + Math.random() * b
```

Returns a random number between `a` and `a + b`, excluding `a + b`.

Practice

Write a method that

* receives one input: radius
* returns two results: area and perimeter

The idea:

- a method can only return one item, but this item can be an object– objects may contain several values (e.g. arrays)
- in the question above, you can return the two results as a 2-element array.

- Anonymous Array
- Variable-Length Argument Lists

Anonymous Array

Anonymous arrays have no explicit reference variable.

They are created and and initialized in the same line

```
new dataType[]{literal0,literal1,…,literalk}
```

The must be used as part of another statement. For example,

- As an argument

```
printArray( new int[]{3, 1, 2, 6} );
```

- In a for-each loop

```
for (int item: new int[]{3, 1, 2, 6} )
        System.out.println(item);
```

Variable-Length Argument Lists

You can pass a variable number of arguments of the same type to a method.

```
returnType amethod (type... parameterName)
```

Some rules…

- **Only one variable-length parameter** may be specified.
- This parameter must **be the last parameter**.
- You can **pass** an **array** OR a **variable number of arguments** to a variable-length parameter.

Java treats a variable-length parameter as an array.

Variable-Length Argument Lists, cont.

```
public class VarArgsDemo {
    public static void main(String[] args) {
        printMax(34, 3, 3, 2, 56.5);
        printMax(new double[] { 1, 2, 3 });
    }

    public static void printMax(double... numbers) {
        if (numbers.length == 0) {
            System.out.println("No argument passed");
            return;
        }
        double result = numbers[0];
        for (int i = 1; i < numbers.length; i++)
            if (numbers[i] > result)
                result = numbers[i];
        System.out.println("The max value is " + result);
    }
}
```

Output:

```
The max value is 56.5
The max value is 3.0
```

Practice

Write a method with this header and test it.

void printRecord(String name, int age, int... grades)

The **Arrays** Class

The **Arrays** Class

The **java.util.Arrays** class contains useful methods for common array operations.

Consider two arrays int[] myList and int[] secondList

`void sort(myList)`

- Sorts myList into **ascending** numerical order.

`int binarySearch(myList, value)`

- Searches myList for the specified value and returns its index.
- The array **must be sorted** prior to making this call.

`String toString(myList)`

- Returns a string representation of the contents of myList.

`Boolean equals(myList, secondList)`

- Returns **true** if the two arrays are strictly equal, and returns false otherwise. Two arrays are strictly equal if their corresponding elements are the same.

The Arrays Class, cont.

```
public static void main(String[] args) {
    int myList[] = {4, 1, 5, 11, 9, 3, 10, 8, 2};
    int secondList[] = {4, 1, 5, 11, 9, 3, 10, 8, 2};
    if(Arrays.equals(myList, secondList))
        System.out.println("myList is strictly equal to secondList.");
    else
        System.out.println("myList is not strictly equal to secondList");
    System.out.print("myList prior sorting: ");
    System.out.println(Arrays.toString(myList));
    Arrays.sort(myList);
    System.out.print("myList after sorting: ");
    System.out.println(Arrays.toString(myList));
    System.out.print("The index of number 11 in the sorted myList: ");
    System.out.println(Arrays.binarySearch(myList, 11));
}
```

Output:

```
myList is strictly equal to secondList.
myList prior sorting: [4, 1, 5, 11, 9, 3, 10, 8, 2]
myList after sorting: [1, 2, 3, 4, 5, 8, 9, 10, 11]
The index of number 11 in the sorted myList: 8
```

UBC OKANAGAN

COSC 111
Computer Programming I

Multidimensional Arrays

Dr. Firas Moosvi

Acknowledgement: Slides (provided by Dr. Abdallah Mohammed) mainly rely on the materials prepared by Y. D. Liang for the textbook "Introduction to Java Programming, 10th Ed., Pearson Edu. Inc.".

Motivation

Data in a table or a matrix can be represented using a two-dimensional array

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

Motivations

In Java, …

```
double[][] distances = {
  {0, 983, 787, 714, 1375, 967, 1087},
  {983, 0, 214, 1102, 1763, 1723, 1842},
  {787, 214, 0, 888, 1549, 1548, 1627},
  {714, 1102, 888, 0, 661, 781, 810},
  {1375, 1763, 1549, 661, 0, 1426, 1187},
  {967, 1723, 1548, 781, 1426, 0, 239},
  {1087, 1842, 1627, 810, 1187, 239, 0},
};
```

Declaration, Creation, and Initialization

Declaring & Creating 2D Arrays

Declare and create a 2D array in ONE statement

```
int[][] x = new x[5][10];
```

Declare and create a 2D array in two statements

```
int[][] x;
x = new int[5][10]; //5 rows, 10 columns
```

Declare, create, and initialize a 2D array in ONE statement

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

Same as

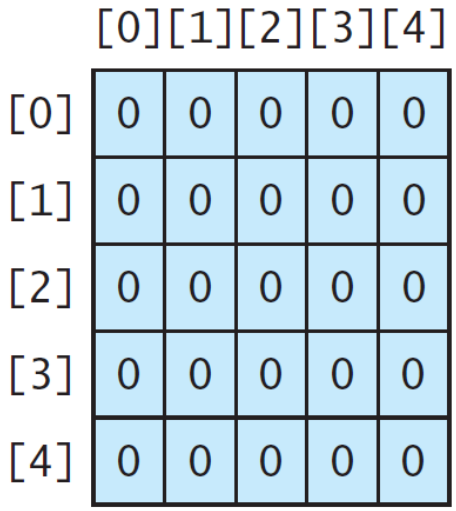
```
int[][] array = new int[4][3];

array[0][0] = 1;   array[0][1] = 2;   array[0][2] = 3;
array[1][0] = 4;   array[1][1] = 5;   array[1][2] = 6;
array[2][0] = 7;   array[2][1] = 8;   array[2][2] = 9;
array[3][0] = 10;  array[3][1] = 11;  array[3][2] = 12;
```

An element in a 2D array is accessed through a **row** and **column** index.

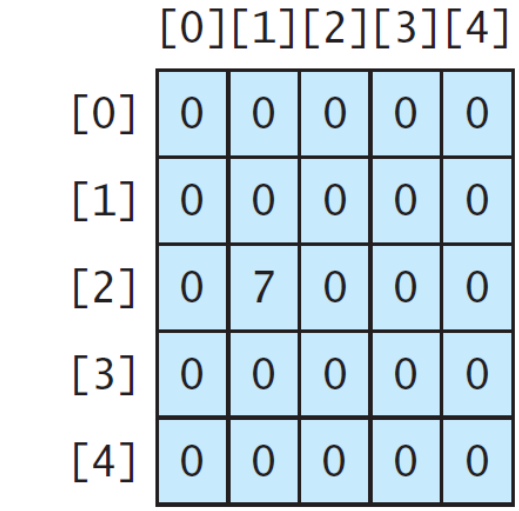
Declaring & Creating 2D Arrays, cont.

Examples:



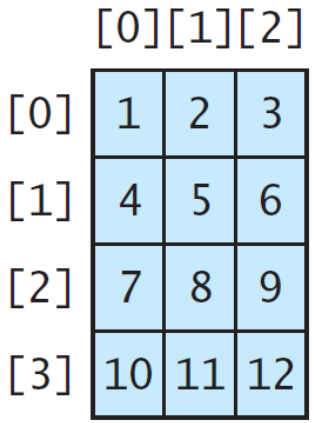
	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

```
matrix = new int[5][5];
```



	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	7	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

```
matrix[2][1] = 7;
```



	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9
[3]	10	11	12

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

Clicker Question

Which of the following correctly declares a 4x5 array?

A.
```
int[] arr = new int[4,5];
```

B.
```
int[] arr;
arr = new int[4][5];
```

C.
```
int[][] arr = new int[4,5];
```

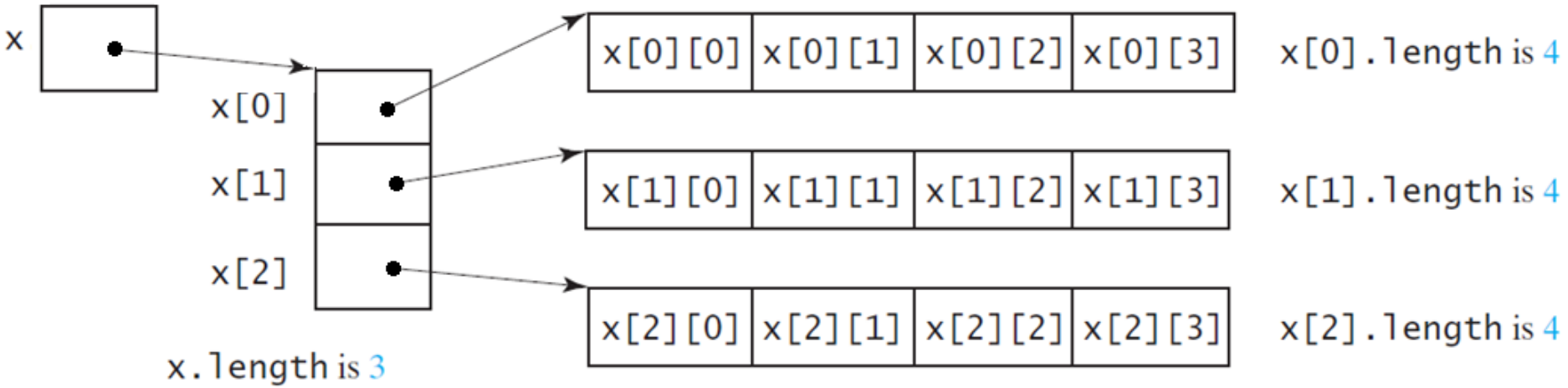
D.
```
int[][] arr;
arr = new int[4][5];
```

E. None of the above

How Java Implements 2D Arrays

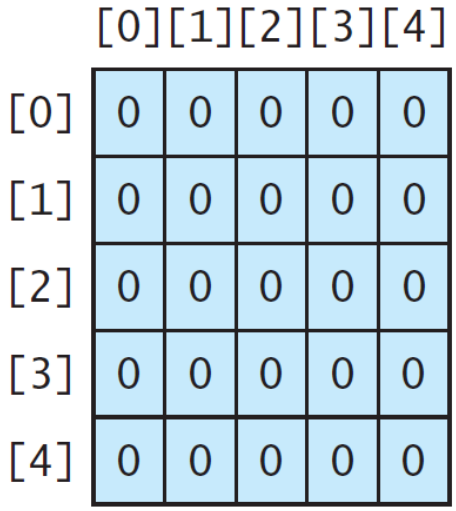
In Java, a two-dimensional array is actually an array in which each element is a one-dimensional array.

```
int[][] x = new int[3][4];
```

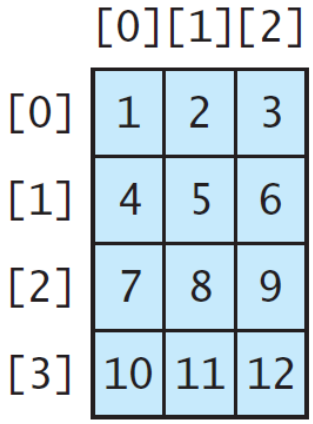


Practice

What is the length of each of the following 2D arrays?




```
matrix = new int[5][5];
```




```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

matrix.length returns 5 (# of *rows*)

matrix[0].length? 5 (# of *columns within the 1st row*)

array.length? 4

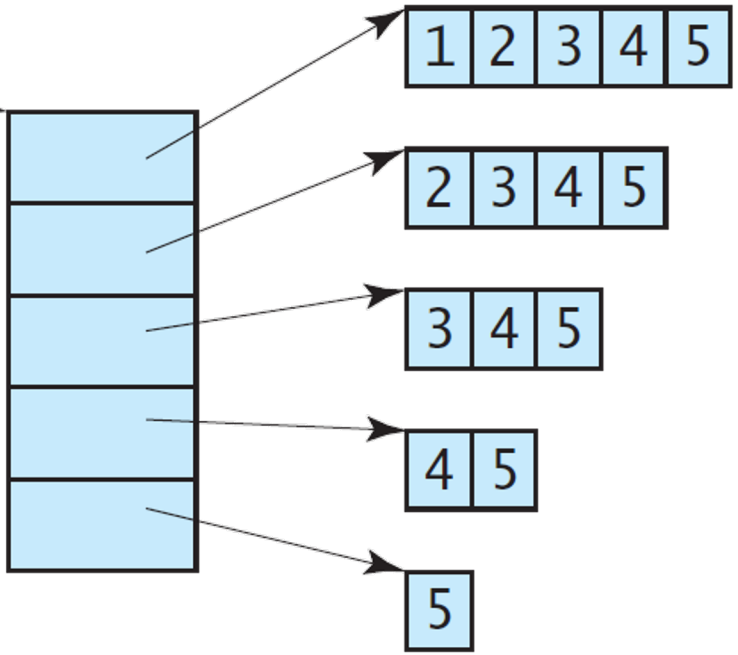
array[0].length? 3

Ragged Arrays

A ragged array is the one in which the **rows can have different lengths**..

- Remember that each row in a two-dimensional array is itself an array.
- **For example,**

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



What is the length of

triangleArray[0], → 5

triangleArray[1], → 4

… ,

triangleArray[4]. → 1

Ragged Arrays, cont'd

The following code declares a ragged array without initializing it

```
int[][] arr = new int[3][];
arr[0] = new int[3];
arr[1] = new int[10];
arr[2] = new int[5];
```

second dimension is omitted as it will be different for each row

Clicker Question

What is the value of `i`?

```
int[][] arr = new int[5][3];
int i = arr[0].length;
```

A. error

B. 0

C. 3

D. 5

E. 15

Clicker Question

Assume `int[][] arr = new int[2][3];`

Which of the following assigns the correct number of rows to **`rows`** variable?

A. `rows = arr.length;`

B. `rows = arr[0].length;`

C. `rows = arr[1].length;`

D. Either B or C

E. All of the above

Clicker Question

Assume `int[][] arr = new int[2][3];`

Which of the following assigns the correct number of columns to **`cols`** variable?

A. `cols = arr.length;`

B. `cols = arr[0].length;`

C. `cols = arr[1].length;`

D. Either B or C

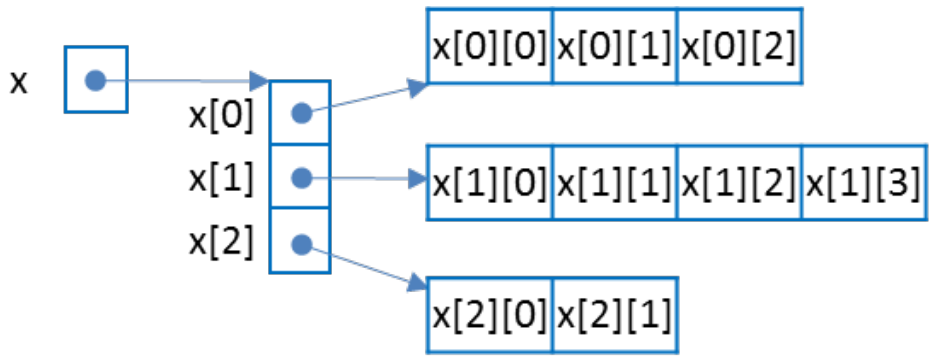
E. All of the above

Processing 2D Arrays

Processing 2D Array

Similarly to 1D arrays, you may use for loops for accessing 2D arrays. A common syntax to process **all elements evenly** is as follows:

```
for (int r = 0; r < x.length; r++){
    for (int c = 0; c < x[r].length; c++){
        //statements that are applied to all elements evenly
    }
}
```



Examples

Initializing a 2D array with random values:

```
for (int r = 0; r < x.length; r++)
      for (int c = 0; c < x[r].length; c++)
         x[r][c] = (int)(Math.random()*1000);
```

Initializing arrays with input values

```
Scanner input = new Scanner(System.in);
System.out.println("Enter " + x.length +
        " rows and " + x[0].length +" columns: ");

for (int r = 0; r < x.length; r++)
     for (int c = 0; c < x[r].length; c++)
          x[r][c] = input.nextInt();
```

Examples, cont.

Printing 2D Arrays:

```
for (int r = 0; r < x.length; r++) {
    for (int c = 0; c < x[r].length; c++)
        System.out.print(x[r][c] + " ");
    System.out.println();
}
```

Finding the sum of all elements:

```
int total = 0;
for (int r = 0; r < x.length; r++)
    for (int c = 0; c < x[r].length; c++)
        total += x[r][c];
```

Examples, cont.

Summing all elements by row:

```
for (int r = 0; r < x.length; r++) {
    int rowTotal = 0;
    for (int c = 0; c < x[r].length; r++)
        rowTotal += x[r][c];
    System.out.println("Sum for row#"+r+" is "+ rowTotal);
}
```

Summing all elements by column:

```
for (int c = 0; c < x[0].length; c++) {
    int colTotal = 0;
    for (int r = 0; r < x.length; r++)
        colTotal += x[r][c];
    System.out.println("Sum for col #"+c+" is "+ colTotal);
}
```

Practice

For a given 2D array, write code to:

- Find the row that has the largest sum.
- Find the smallest index of the largest element.
- Randomly shuffle array's elements.

Practice

Write a program that grades multiple-choice test.

Assume the following data is given and you are required to display the grade for each student.

Students' Answers

	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

Key to the Questions:

	0	1	2	3	4	5	6	7	8	9
Key	D	B	D	C	C	D	A	E	A	D

Practice, cont.

Algorithm:

1. Store data for students' answer and key in arrays.

2. For each student (rows),
   a. initialize a counter for counting student's correct answers.
   b. For each question (columns),
      - if student's answer is correct, increment the counter
   c. display the count of correct answers for that student.

3. End the program

Practice, cont.

```
char[][] answers = {
                   {'a', 'b', 'a', 'a', 'a', 'a', 'c'}, //student0
                   {'c', 'c', 'd', 'b', 'a', 'c', 'd'}, //student1
                   {'d', 'b', 'c', 'a', 'a', 'd', 'c'}, //student2
                   {'a', 'c', 'c', 'a', 'b', 'a', 'c'}  //student3
               };
char[] keys =    {'a', 'c', 'c', 'a', 'b', 'a', 'c' };

//for each student, compute the score
for (int student = 0; student < answers.length; student++) {
    int score = 0;
    //check each question and increment score if correct
    for (int question = 0; question < keys.length; question++) {
        if (answers[student][question] == keys[question])
            score++;
    }
    System.out.printf("Student%d's score: %d\n", student, score);
}
```

Special cases

Special cases

In some cases, we don't use 2 nested for loops to iterate over 2D arrays. In other cases, the for loops do not cover the whole range of indexes.

The following slides have two examples:

- Example 1: No inner for loop.
- Example 2: Two for loops that don't follow the standard format

Using ONE for loop

Code to read the price and quantity of several items.

	price	quantity
Item 0		
Item 1		
Item 2		
Item 3		

```
int[][] table = new int[4][2];          //4 rows, 2 cols
for (int item = 0; item < 2; item++) { //for each item
      System.out.printf("Enter the price of item#%d: ",item);
      table[item][0] = input.nextInt();
      System.out.printf("Enter the number of items: ");
      table[item][1] = input.nextInt();
}
```

Two for loops that don't follow the standard format

Code to read the ID and 3 grades for several students.

	ID	Grade 1	Grade 2	Grade 3
Student 0				
Student 1				
Student 2				

```
int[][] grades = new int[3][4];                    // 3 students, 4 entries
for (int student = 0; student < grades.length; student++) {
        System.out.printf("Enter the ID of Student#%d: ", student+1);
        grades[student][0] = input.nextInt();
        for (int assignment = 1; assignment <= 3; assignment++) {
                System.out.printf("Enter grade for A%d: ", assignment);
                grades[student][assignment] = input.nextInt();
        }
}
```

2-D Arrays to/from Methods

Multidimensional Arrays & Methods

Same rules studied before (in Chapter 7) apply here!

- Passing 2-D Arrays to Methods:
  - When passing a 2-D array to a method, the reference of the array is passed to the method.
  - You have to have method parameters declared of the same type and dimension of the arguments.

- Returning 2-D Arrays to Methods:
  - When a method returns an array **the reference of the array is returned**.

Example: Method to print 2D arrays

```
public static void main(String[] args) {
      int[][] x = { { 3, 4, 6, 7 },
                    { 1, 2, 6, 2 } };
      print2DArray(x);
}

public static void print2DArray(int[][] arr) {
      for (int r = 0; r < arr.length; r++)
            for (int c = 0; c < arr[r].length; c++)
                  System.out.print(arr[r][c] + " ");
            System.out.println();
      }
}
```

Example: Method to Return a New Array

```
public static void main(String[] args) {
      double[][] x = getRandom2DArray(3,5);
}

public static double[][] getRandom2DArray(int n, int m) {
      double[][] temp = new double[n][m];
      for (int r = 0; r < n; r++)
            for (int c = 0; c < m; c++)
                  temp[r][c] = Math.random();
      return temp;
}
```

Example…Both methods together

```
public static void main(String[] args) {
    double[][] x = getRandom2DArray(3,5);
    print2DArray(x);
}

public static double[][] getRandom2DArray(int n, int m) {
    double[][] temp = new double[n][m];
    for (int r = 0; r < n; r++)
        for (int c = 0; c < m; c++)
            temp[r][c] = Math.random();
    return temp;
}

public static void print2DArray(double[][] arr) {
    for (int r = 0; r < arr.length; r++)
        for (int c = 0; c < arr[r].length; c++
            System.out.printf("%-6.2f", arr[r][c]);
    System.out.println()
}
```

Clicker Question

What is the value of **arr** array?

```
public static void main(String[] args) {
   int[][] arr = { {1,2,3},
                   {4,5,6} };
   arr = zeros(2,2);
}
public static int[][] zeros(int n, int m) {
   return new int[n][m];
}
```

A. 1 2 3
   4 5 6

B. 0 0 0
   0 0 0

C. 1 2
   4 5

D. 0 0
   0 0

Optional Readings

Multidimensional Arrays

Multidimensional Arrays

If you need to represent n-dimensional data structures, you can create nD arrays.

- A 2D array is an array of 1D arrays
- A 3D array is an array of 2D arrays.
- …

Example:

- `int[] x = new int[10]; //1D array`
- `int[][] y = new int[5][12]; //2D array`
- `int[][][] z = new int[2][7][3]; //3D array`

This defines

- x array of 10 integers, y of 5 by 12 matrix of integers, and z of 2 by 7 by 3 array of integers.

You can then access these elements, for example:

- `y[2][3] = x[0] + z[2][1][5];`

Example: Calculating Total Scores

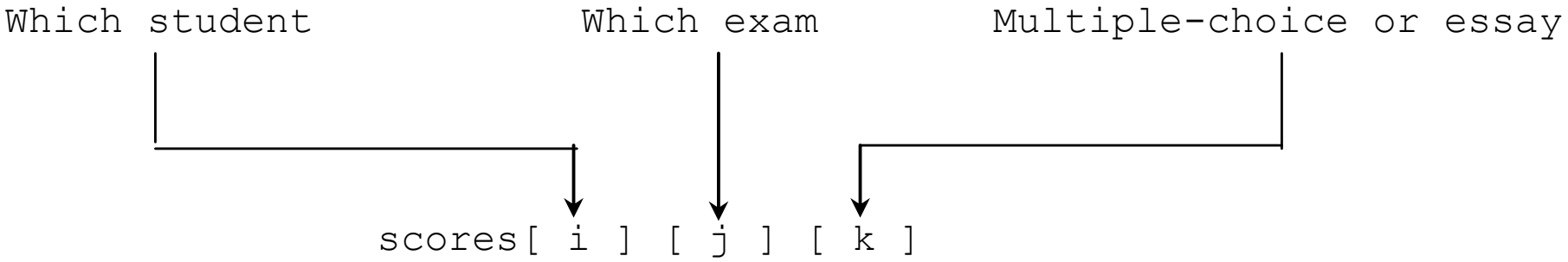
Write a program that calculates **the total score for students** in a class. Suppose the scores are stored in a 3D array named scores.

- The first index in scores refers to a student,
- the second refers to an exam, and
- the third refers to the part of the exam.

Suppose there are 7 students, 5 exams, and each exam has two parts--the multiple-choice part and the programming part.

- e.g., for the i's student on the j's exam: scores[i][j][0] represents the score on the multiple-choice part, and scores[i][j][1] represents the score on the programming part.

Your program displays the total score for each student.



Problem: Calculating Total Scores, cont.

```
double[][][] scores = {
  { {7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5} },
  { {4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5} },
  { {6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5} },
  { {6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5} },
  { {8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5} },
  { {9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5} }
};
```

scores[0][1][0] refers to the multiple-choice score for the first student's second exam, which is **9.0**.

scores[0][1][1] refers to the essay score for the first student's second exam, which is **22.5**.

```
double[][][] scores={
  { {7.5, 20.5},
    {9.0, 22.5},
    {15,  33.5},
    {13,  21.5},
    {15,  12.5} },
  { {4.5, 21.5},
    {9.0, 22.5},
    {15, 34.5},
    {12, 20.5},
    {14,   9.5} },
…
```

Problem: Calculating Total Scores, cont.

Algorithm:

- Store data for students' answer and key in arrays.

1. for each student,
   a. initialize a variable, *totalScore*, for summing the student's score.
   b. For each exam,
      For each question,
      add the question's grade to *totalScore*
   c. display the count of correct answers for that student.

2. End the program

Problem: Calculating Total Scores, cont.

```
public class TotalScore {
    public static void main(String args[]) {
        double[][][] scores = {
          {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
          {{4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
          {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
          {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
          {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
          {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}},
          {{1.5, 29.5}, {6.4, 22.5}, {14, 30.5}, {10, 30.5}, {16, 6.0}}};

        // Calculate and display total score for each student
        for (int i = 0; i < scores.length; i++) {
            double totalScore = 0;
            for (int j = 0; j < scores[i].length; j++)
                for (int k = 0; k < scores[i][j].length; k++)
                    totalScore += scores[i][j][k];

            System.out.println("Student " + i + "'s score is " + totalScore);
        }
    }
}
```