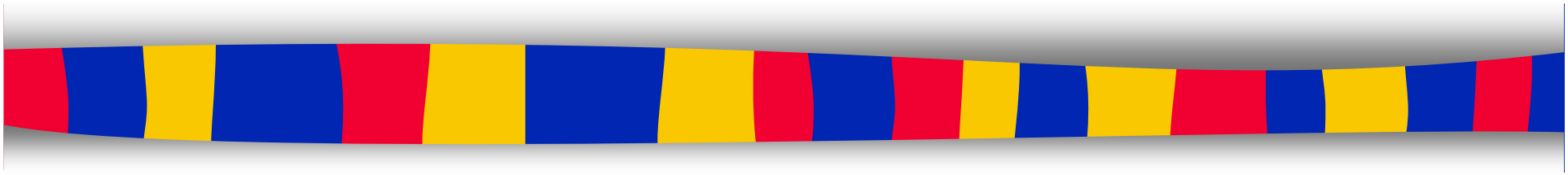
COMP-248
Object Oriented Programming I



Sorting and 2D Arrays

By Emad Shihab, PhD, Fall 2015,
Parts of the slides are taken from Prof. L. Kosseim
Adapted for Section EE by S. Ghaderpanah, Fall 2015

Next:

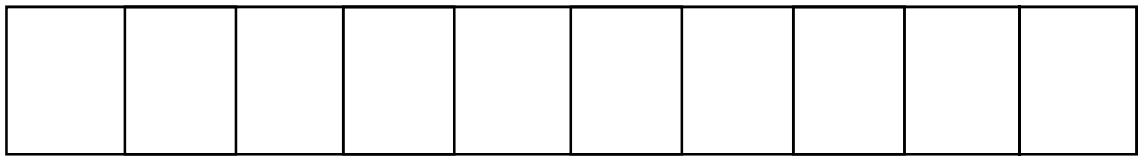
1. Arrays
2. **Sorting**
3. "For each" loop
4. Multidimensional arrays

Arrays

An *array* is an ordered list of elements of the same type

The entire array has a single name

Each value has a numeric *index*



This array holds 10 values that are indexed from 0 to 9

An array of size n is indexed from 0 to n-1

Example 2: Difference from max

```
Enter 5 scores:
80 99.9 75 100 85.5
The highest score is 100
The scores are:
80.0 differs from max by 20
…
```

Variables needed?

Algorithm?

Code

```
Scanner keyboard = new Scanner(System.in);
double[] score = new double[5];
int index;
double max;

System.out.println("Enter " + score.length + " scores:");
score[0] = keyboard.nextDouble( );
max = score[0];
for (index = 1; index < score.length; index++)
{
    score[index] = keyboard.nextDouble( );
    if (score[index] > max)
        max = score[index];
//max is the largest of the values score[0],..., score[index].
}

System.out.println("The highest score is " + max);
System.out.println("The scores are:");
for (index = 0; index < score.length; index++)
    System.out.println(score[index] + " differs from max by "
                       + (max - score[index]));
```

Next:

1. Arrays
2. **Sorting**
3. For each loop
4. Multidimensional arrays

2- Sorting (p. 382)

Sorting is the process of arranging a list of items in a particular order

There are many algorithms for sorting a list of items

some are more efficient, some are more intuitive

- Selection Sort
- Insertion Sort
- Bubble Sort
- Quicksort
- …

Selection Sort

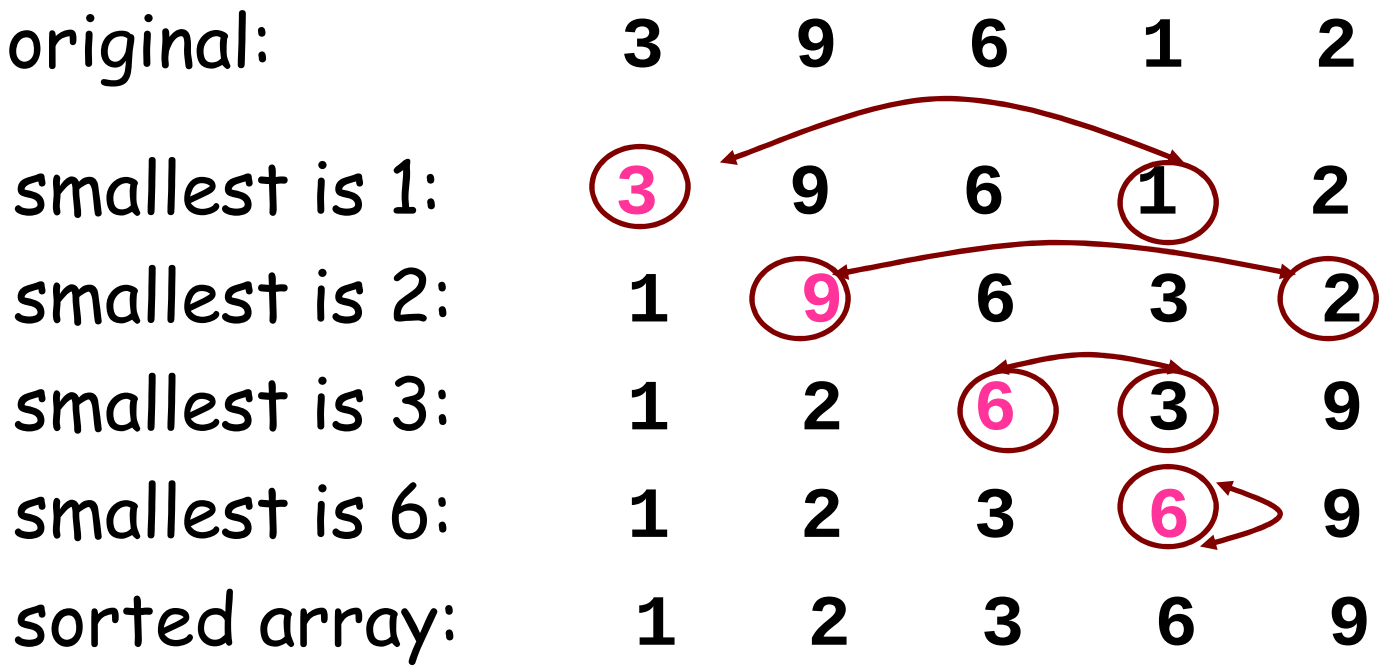
The approach:

- -select a value and put it in its final place into the list
- -repeat for all other values

In more detail:

- -find the smallest value in the array
- -switch it with the value in the first position
- -find the next smallest value in the array
- -switch it with the value in the second position
- -repeat until all values are in their proper places

Example: Selection sort



Example: SelectionSortDemo

```
double[] numbers = new double[20];  // the array to be sorted

// fill the array
for (int i=0; i<numbers.length; i++)
   numbers[i] = keyboard.nextDouble();

// sort the array
int indexOfMin;
double temp;
// for every element (except the last)
for (int index = 0; index < numbers.length-1; index++)
{
   indexOfMin = index;
   // find the index of the smallest element between index and the last
   for (int scan = index+1; scan < numbers.length; scan++)
      if (numbers[scan] < numbers[indexOfMin])
         indexOfMin = scan;

    // Swap the smallest element with the one at position 'index'
    temp = numbers[indexOfMin];
    numbers[indexOfMin] = numbers[index];
    numbers[index] = temp;
}
```

In this chapter, we will see…

1. Arrays
2. Sorting
3. **"For each" loop**
4. Multidimensional arrays

3- The "for each" Loop (p. 370)

An ordinary **for** loop --> manual traversal of an array

An enhanced **for** loop can traverse automatically through the elements of an array

Syntax:

```
for (ArrayBaseType VariableName : ArrayName)
   Statement
```

Example

Regular for:

```
for (int i = 0; i < a.length; i++)
  a[i] = 0.0;
```

Foreach loop

```
for (double element : a)
  System.out.println(element);
```

Example: SelectionSortDemo

```
…..
// sort the array
int indexOfMin;
double temp;
// for every element (except the last)
for (int index = 0; index < numbers.length-1; index++)
{
   indexOfMin = index;
   // find the index of the smallest element between index and the last
   for (int scan = index+1; scan < numbers.length; scan++)
      if (numbers[scan] < numbers[indexOfMin])
         indexOfMin = scan;

    // Swap the smallest element with the one at position 'index'
    temp = numbers[indexOfMin];
    numbers[indexOfMin] = numbers[index];
    numbers[index] = temp;
}
for (double element: numbers)
{
    System.out.println(element);
}
```

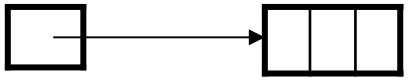
Next:

1. Arrays
2. Sorting
3. "For each" loop
4. **Multidimensional arrays**

4- Multidimensional arrays(p. 393)

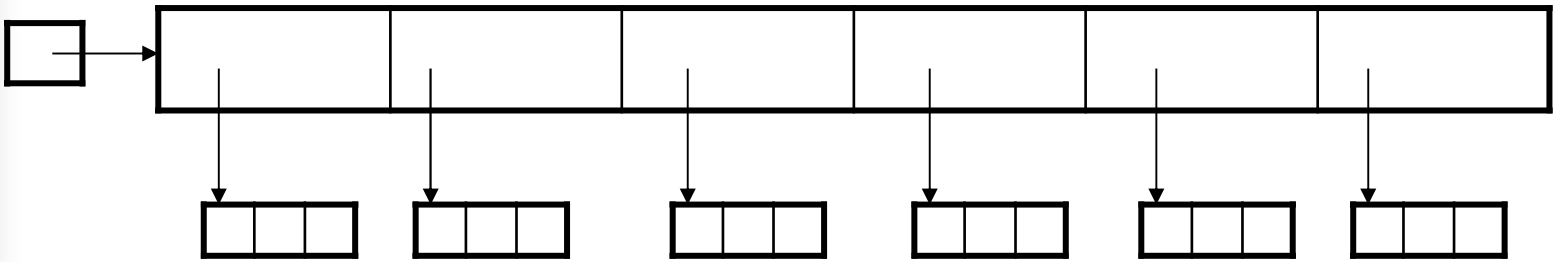
A one-dimensional array

stores a list of elements of simple type (primitive or reference)



A two-dimensional array

stores a list of elements, where each element is a 1-D array of simple type

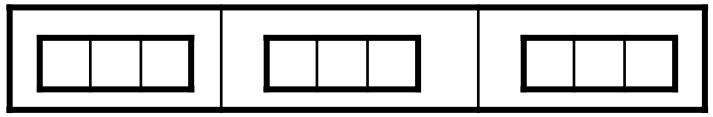


Multidimensional arrays

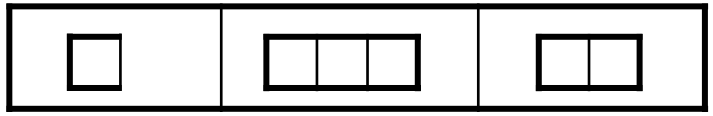
a 2-D array is really a 1-D array of references to 1-D arrays

so the arrays within one dimension can be of different lengths (called *ragged arrays)*

not :



but:



Two-dimensional arrays

Declaration:

```
double[] student = new double[5]; // 1-D 5 tests for 1 student
double[][] section = new double[5][80]; // 2-D 80 students per section
double[][][] course = new double[3][5][80]; // 3-D 5 sections per course
```

Access to an element

```
value = section[3][5]
```

Just checking ….

Given the declaration:

```
double[][][] costOfGoods = new double [8][2][7];
```

how many elements does **costOfGoods** contain?

A. 8
B. 70
C. 72
D. 112
E. 17

Just checking ….

Given the declarations:

```
double[] x = new double[300];
double[][] y = new double[75][4];
double[] z = new double[79];
```

which of the following statements is true?

A. **x** has more elements than **y**.

B. **y** has more elements than **x**.

C. **y** and **z** have the same number of elements.

D. **x** and **y** have the same number of elements.

E. A and C above

Length with multidimensional arrays

```
char[][] page = new char[30][100];
```

`length` does not give the total number of indexed variables

- `page.length` is equal to 30
- `page[0].length` is equal to 100

```
int row, col;
for (row = 0; row < page.length; row++)
  for (col = 0; col < page[row].length; col++)
    page[row][col] = 'Z';
```