

# ARRAYS



# Chapter Goals

- To collect elements using arrays and array lists
- To use the enhanced for loop for traversing arrays and array lists
- To learn common algorithms for processing arrays and array lists
- To work with two-dimensional arrays

In this chapter, you will learn about arrays, array lists, and common algorithms for processing them.

# Contents

- Arrays
- The Enhanced for Loop
- Common Array Algorithms
- Using Arrays with Methods
- Problem Solving:
  - Adapting Algorithms
  - Discovering Algorithms by Manipulating Physical Objects
- Two-Dimensional Arrays
- Array Lists



# 6.1 Arrays

- A Computer Program often needs to store a list of values and then process them
- For example, if you had this list of values, how many variables would you need?
  - double input1, input2, input3....
- Arrays to the rescue!

32
54
67.5
29
35
80
115
44.5
100
65

An array collects sequences of values of the same type.

# Declaring an Array

- Declaring an array is a two step process

1) `double[] values;` // declare array variable

2) `values = new double[10];` // initialize array

values =

values =

double[]

0

0

0

0

0

0

0

0

0

0

You cannot use the array until you tell the compiler the size of the array in step 2.

Declare the array variable

Initialize it with an array

# Declaring an Array (Step 1)

- Make a named 'list' with the following parts:

Type	Square Braces	Array name	semicolon
double	[ ]	values	;

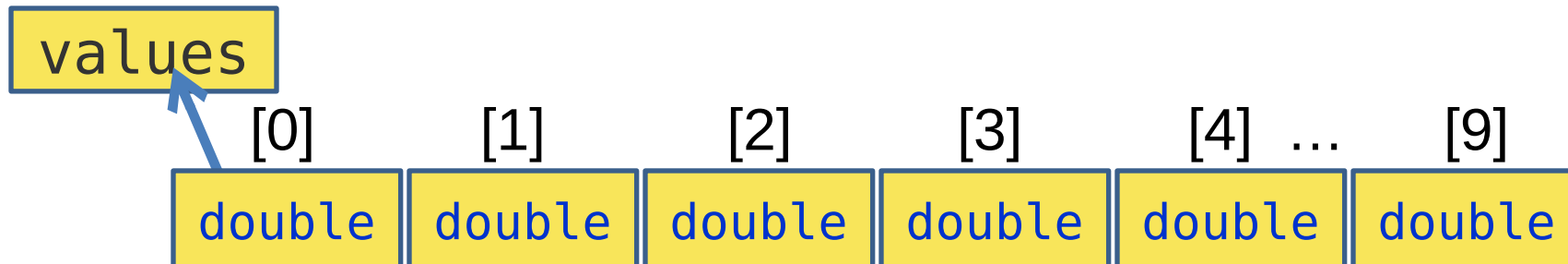
- You are declaring that
  - There is an array named **values**
  - The elements inside are of type **double**
  - You have not (YET) declared how many elements are in inside
- Other Rules:
  - Arrays can be declared anywhere you can declare a variable
  - Do not use 'reserved' words or already used names

# Declaring an Array (Step 2)

- Reserve memory for all of the elements:

Array name	Keyword	Type	Size	semicolon	
values	=	new	double	[10]	;

- You are reserving memory for:
  - The array named **values** needs storage for **[10]** elements the size of type **double**
- You are also setting up the array variable
- Now the compiler knows how many elements there are
  - You cannot change the size after you declare it!



# One Line Array Declaration

- Declare and Create on the same line:

Type	Braces	Array name	Keyword	Type	Size	semi
double	[]	values	=	new		

```
double[10] ;
```

- You are declaring that
  - There is an array named `values`
  - The elements inside are of type `double`
- You are reserving memory for the array
  - Needs storage for `[10]` elements the size of type `double`
- You are also setting up the array variable



# Declaring and Initializing an Array

- You can declare and set the initial contents of all elements by:

Type	Braces	Array name	contents list	semi
<code>int</code>	<code>[ ]</code>	<code>primes</code>	<code>= { 2, 3, 5, 7 }</code>	<code>;</code>

- You are declaring that
  - There is an array named `primes`
  - The elements inside are of type `int`
  - Reserve space for four elements
    - The compiler counts them for you!
  - Set initial values to 2, 3, 5, and 7
  - Note the curly braces around the contents list

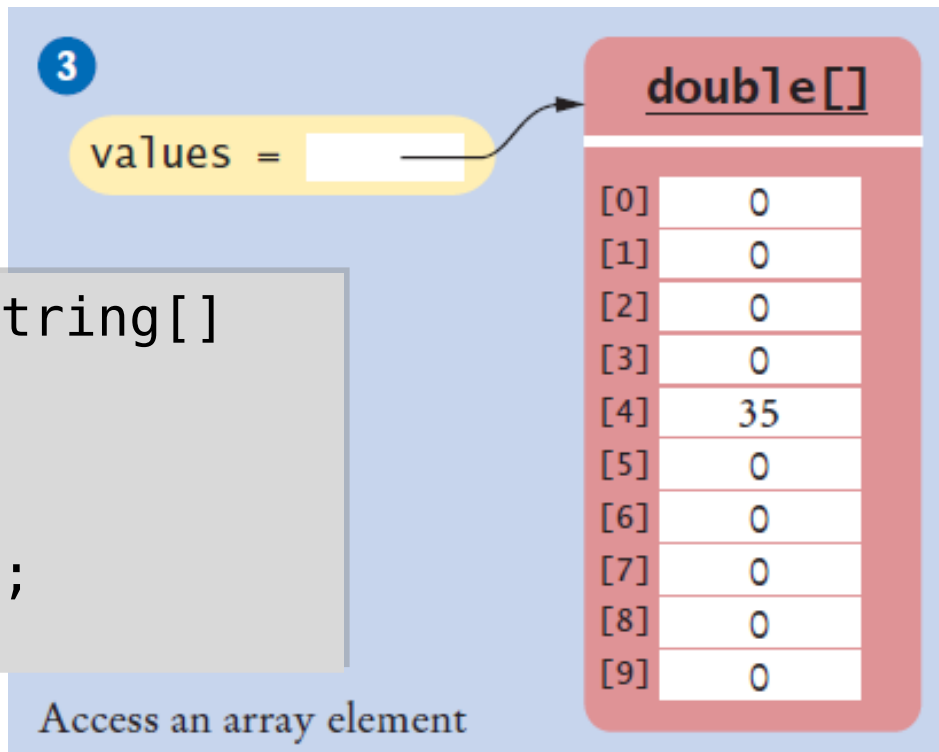
# Accessing Array Elements

- Each element is numbered
  - We call this the *index*
  - Access an element by:
    - Name of the array
    - Index number

`values[i]`

Elements in the array `values` are accessed by an integer index `i`, using the notation `values[i]`.

```
public static void main(String[]  
    args)  
{  
    double values[];  
    values = new double[10];  
    values[4] = 35;  
}
```



# Syntax 6.1: Array

- To declare an array, specify the:
  - Array variable name
  - Element Type
  - Length (number of elements)

Diagram illustrating array syntax with annotations:

```
double[] values = new double[10];
```

Annotations for the first line:

- Type of array variable** points to `double[]`.
- Name of array variable** points to `values`.
- Element type** points to `double` in `new double[10]`.
- Length** points to `10` in `new double[10]`.

```
double[] moreValues = { 32, 54, 67.5, 29, 35 };
```

Annotation for the second line:

- List of initial values** (in a yellow oval) points to the list `{ 32, 54, 67.5, 29, 35 }`.

Use brackets to access an element.

```
values[i] = 0;
```

Annotation for the third line:

- The index must be  $\geq 0$  and  $<$  the length of the array.** (with a spider icon) points to `i`.
- See page 255.** points to the same line.

# Array Index Numbers

- Array index numbers start at 0
  - The rest are positive integers
- An 10 element array has indexes 0 through 9
  - **There is NO element 10!**

The first element is at index 0:

```
public static void main(String[]  
    args)  
{  
    double values[];  
    values = new double[10];  
}
```

The last element is at index 9:

<u>double[]</u>	
[0]	0
[1]	0
[2]	0
[3]	0
[4]	35
[5]	0
[6]	0
[7]	0
[8]	0
[9]	0

# Array Bounds Checking


- An array knows how many elements it can hold
  - `values.length` is the size of the array named `values`
  - It is an integer value (index of the last element + 1)
- Use this to range check and prevent bounds errors

```
public static void main(String[] args)
{
    int i = 10, value = 34;
    double values[];
    values = new double[10];
    if (0 <= i && i < values.length)    // length is 10
    {
        value[i] = value;
    }
}
```

Strings and arrays use different syntax to find their length:  
Strings: `name.length()`  
Arrays: `values.length`

# Summary: Declaring Arrays

**Table 1** Declaring Arrays

<pre>int[] numbers = new int[10];</pre>	An array of ten integers. All elements are initialized with zero.
<pre>final int LENGTH = 10; int[] numbers = new int[LENGTH];</pre>	It is a good idea to use a named constant instead of a “magic number”.
<pre>int length = in.nextInt(); double[] data = new double[length];</pre>	The length need not be a constant.
<pre>int[] squares = { 0, 1, 4, 9, 16 };</pre>	An array of five integers, with initial values.
<pre>String[] friends = { “Emily”, “Bob”, “Cindy” };</pre>	An array of three strings.
 <pre>double[] data = new int[10]</pre>	<b>Error:</b> You cannot initialize a double[] variable with an array of type int[].

# Array References

- Make sure you see the difference between the:
  - Array variable: The named 'handle' to the array
  - Array contents: Memory where the values are stored

```
int[] scores = { 10, 9, 7, 4, 5 };
```

Array variable

scores =



Reference

Array contents



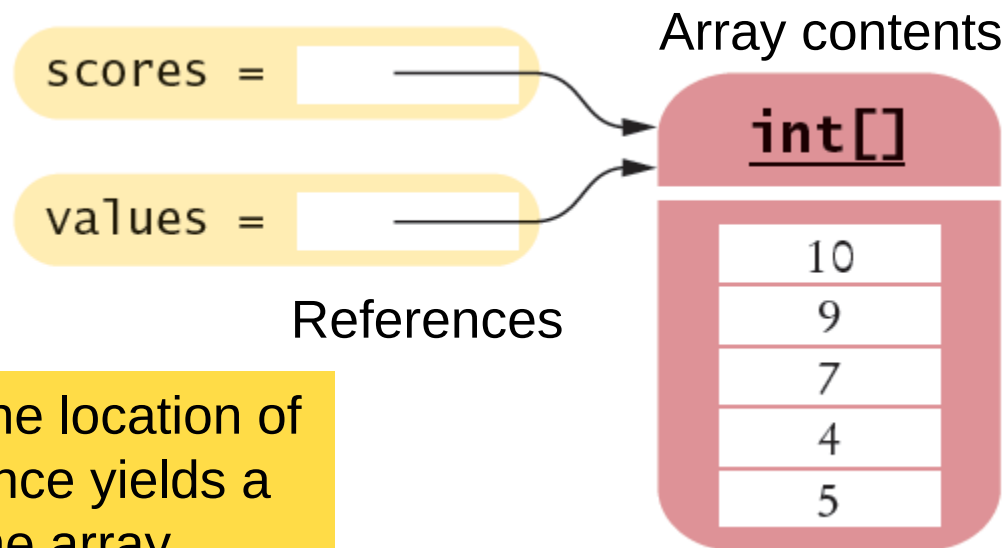
An array variable contains a *reference* to the array contents. The *reference* is the location of the array contents (in memory).

Values

# Array Aliases

- You can make one array reference refer to the same contents of another array reference:

```
int[] scores = { 10, 9, 7, 4, 5 };  
int[] values = scores; // Copying the array  
reference
```



An array variable specifies the location of an array. Copying the reference yields a second reference to the same array.



# Partially-Filled Arrays

- An array cannot change size at run time
  - The programmer may need to guess at the maximum number of elements required
  - It is a good idea to use a constant for the size chosen
  - Use a variable to track how many elements are filled

```
final int LENGTH = 100;
double[] values = new double[LENGTH];
int currentSize = 0;
Scanner in = new Scanner(System.in);
while (in.hasNextDouble())
{
    if (currentSize < values.length)
    {
        values[currentSize] =
            in.nextDouble();
        currentSize++;
    }
}
```

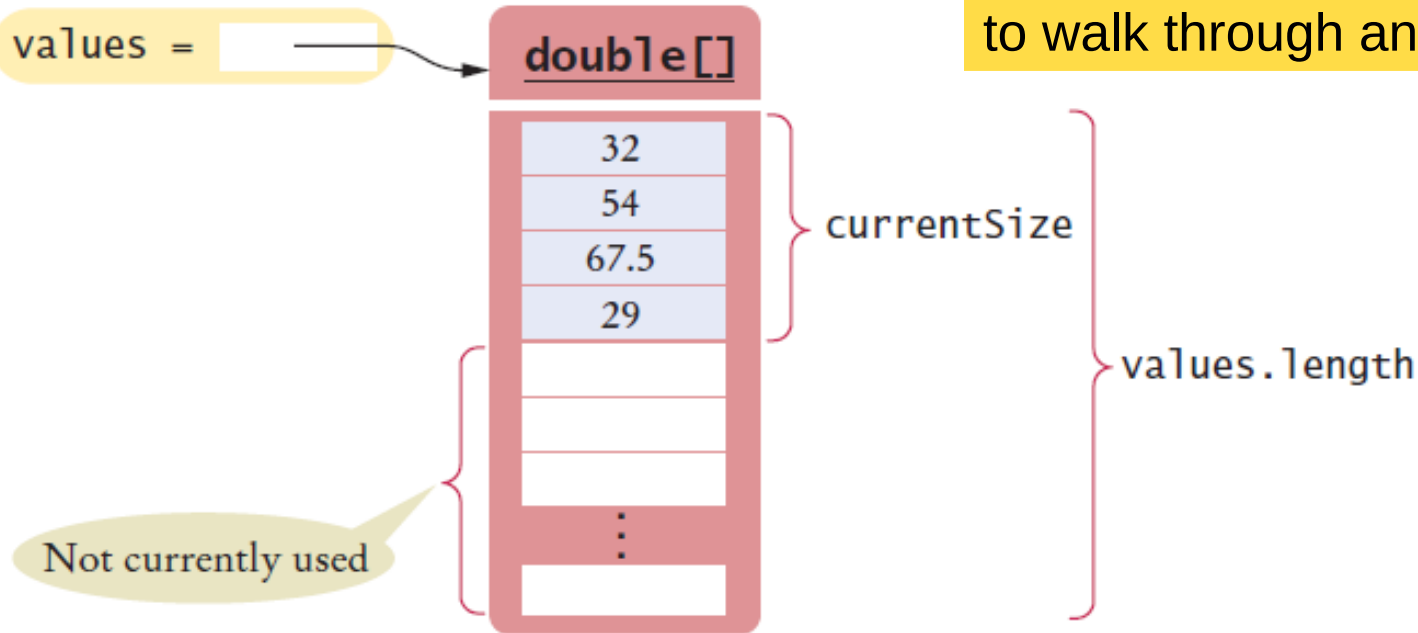
Maintain the number of elements filled using a variable (`currentSize` in this example)

# Walking a Partially Filled Array

- Use **currentSize**, not `values.length` for the last element

```
for (int i = 0; i < currentSize; i++)  
{  
    System.out.println(values[i]);  
}
```

A for loop is a natural choice to walk through an array



# Common Error 6.1



- Array Bounds Errors
  - Accessing a nonexistent element is very common error
  - Array indexing starts at 0
  - Your program will stop at run time

```
public class OutOfBounds
{
    public static void main(String[]
        args)
    {
        double values[];
        values = new double[10];
        values[10] = 100;
    }
}
```

The is no element 10:

<u>double[]</u>	
[0]	0
[1]	0
[2]	0
[3]	0
[4]	35
[5]	0
[6]	0
[7]	0
[8]	0
[9]	0

**java.lang.ArrayIndexOutOfBoundsException: 10  
at OutOfBounds.main(OutOfBounds.java:7)**

## Common Error 6.2



- Uninitialized Arrays

- Don't forget to initialize the array variable!
- The compiler will catch this error

```
double[] values;
```

```
...
```

```
values[0] = 29.95; // Error-values not  
initialized
```

Error: D:\Java\Unitialized.java:7:  
variable values might not have been initialized

1

values =

2

```
double[] values;
```

```
values = new double[10];
```

```
values[0] = 29.95; // No error
```

values =

double[]

0

## 6.2 The Enhanced for Loop

- Using for loops to 'walk' arrays is very common
  - The enhanced for loop simplifies the process
  - Also called the “for each” loop
  - Read this code as:
    - “For each element in the array”
- As the loop proceeds, it will:
  - Access each element in order (0 to length-1)
  - Copy it to the **element variable**
  - Execute loop body
- Not possible to:
  - Change elements
  - Get bounds error

```
double[] values = . . . ;
double total = 0;
for (double element :
    values)
{
    total = total + element;
}
```

## Syntax 6.2: The Enhanced `for` loop

- Use the enhanced “**for**” loop when:
  - You need to access every element in the array
  - You do not need to change any elements of the array

This variable is set in each loop iteration.  
It is only defined inside the loop.

An array

```
for (double element : values)
{
    sum = sum + element;
}
```

These statements  
are executed for each  
element.

The variable  
contains an element,  
not an index.

## 6.3 Common Array Algorithms

- Filling an Array
- Sum and Average Values
- Find the Maximum or Minimum
- Output Elements with Separators
- Linear Search
- Removing an Element
- Inserting an Element
- Swapping Elements
- Copying Arrays
- Reading Input

# Common Algorithms 1 and 2:

## 1) Filling an Array

- Initialize an array to a set of calculated values
- Example: Fill an array with squares of 0 through 10

```
int[] values = new int[11];  
for (int i = 0; i < values.length; i+  
    +)  
{  
    values[i] = i * i;  
}
```

## 2) Sum and Average

- Use enhanced for loop, and make sure not to divide by zero

```
double total = 0, average = 0;  
for (double element : values)  
{  
    total = total + element;  
}  
if (values.length > 0) { average = total /  
    values.length; }
```



# Common Algorithms 3:

```
double largest = values[0];
for (int i = 1; i < values.length;
    i++)
{
    if (values[i] > largest)
    {
        largest = values[i]
    }
}
```

Typical for loop to find maximum

- Maximum and Minimum
  - Set largest to first element
  - Use for or enhanced for loop
  - Use the same logic for minimum

```
double largest = values[0];
for (double element :
    values)
{
    if (element > largest)
        largest = element;
}
```

Enhanced for to find maximum

```
double smallest =
    values[0];
for (double element :
    values)
{
    if (element < smallest)
        smallest = element;
}
```

Enhanced for to find minimum

# Common Algorithms 4:

- Element Separators

- Output all elements with separators between them
- No separator before the first or after the last element

```
for (int i = 0; i < values.length; i++)  
{  
    if (i > 0)  
    {  
        System.out.print(" | ");  
    }  
    System.out.print(values[i]);  
}
```

32 | 54 | 67.5 | 29 | 35



- Handy Array method: `Arrays.toString` [32, 54, 67.5, 29, 35]

```
import java.util.*;  
System.out.println(Arrays.toString(values));
```

# Common Algorithms 5:

- Linear Search

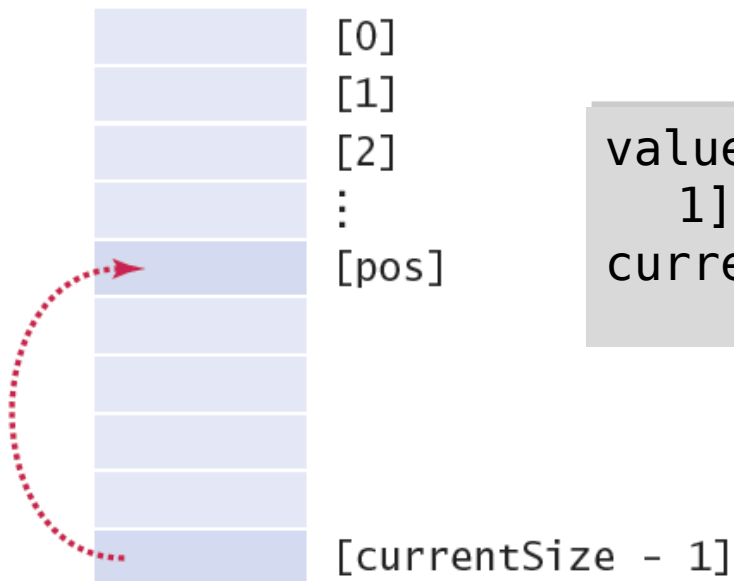
- Search for a specific value in an array
- Start from the beginning (left), stop if/when it is found
- Uses a boolean **found** flag to stop loop if found

```
int searchedValue = 100;  int pos = 0;
boolean found = false;
while (pos < values.length && !found)
{
    if (values[pos] == searchedValue)
    {
        found = true;
    }
    else
    {
        pos++;
    }
    if (found)
    {
        System.out.println("Found at position: " + pos);
    }
    else { System.out.println("Not found");
}
```

Compound condition to prevent bounds error if value not found.

# Common Algorithms 6a:

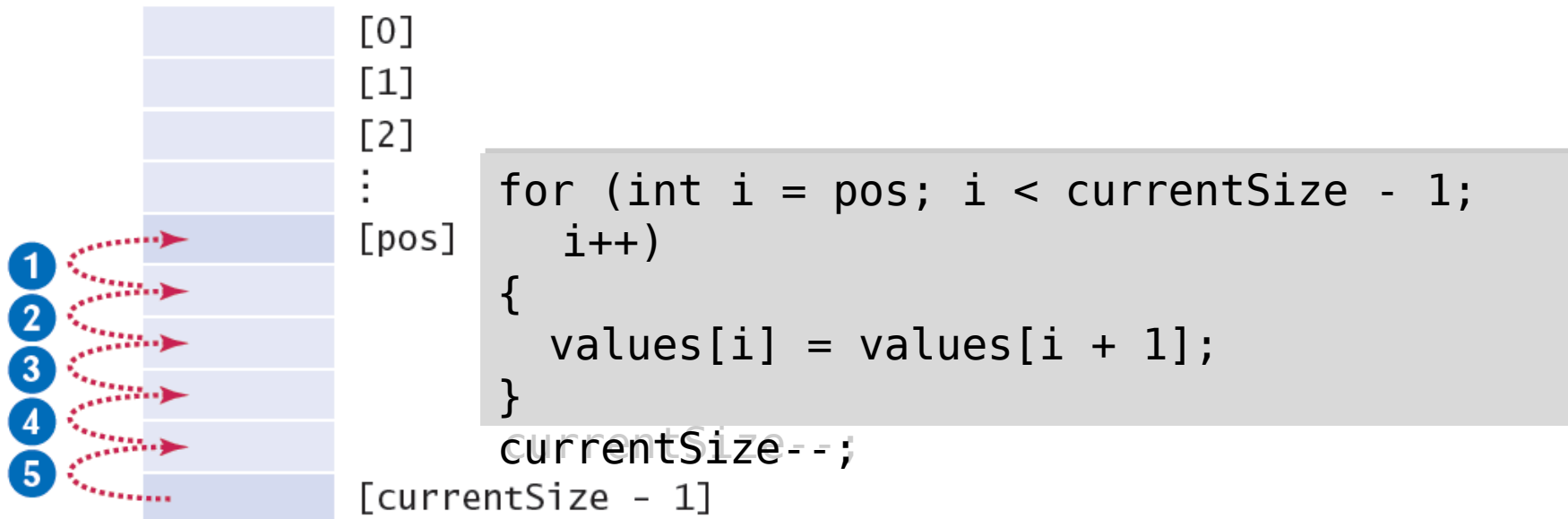
- Removing an element (at a given position)
  - Requires tracking the 'current size' (# of valid elements)
  - But don't leave a 'hole' in the array!
  - Solution depends on if you have to maintain 'order'
    - If not, find the last valid element, copy over position, update size



```
values[pos] = values[currentSize - 1];  
currentSize--;
```

# Common Algorithms 6b:

- Removing an element and maintaining order
  - Requires tracking the 'current size' (# of valid elements)
  - But don't leave a 'hole' in the array!
  - Solution depends on if you have to maintain 'order'
    - If so, move all of the valid elements after 'pos' up one spot, update size

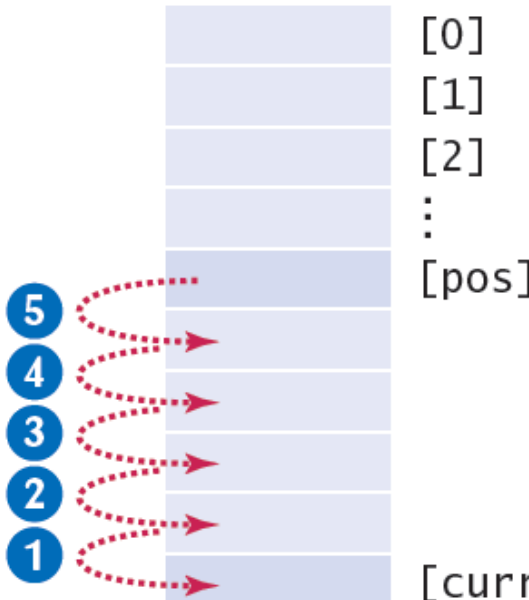


# Common Algorithms 7:

- Inserting an Element

- Solution depends on if you have to maintain 'order'

- If not, just add it to the end and update the size
- If so, find the right spot for the new element, move all of the valid elements after 'pos' down one spot, insert the new element, and update size

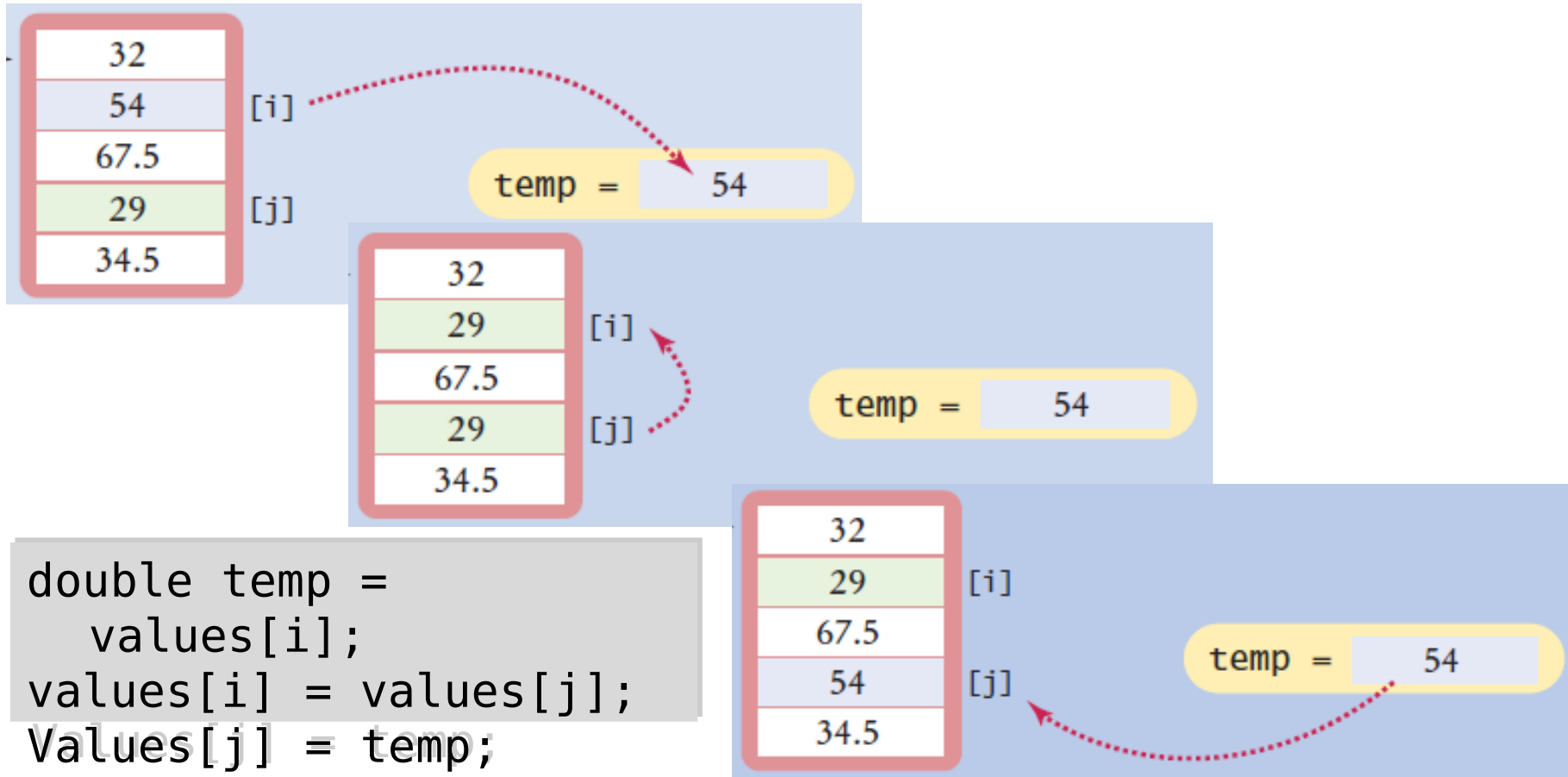


The diagram shows a vertical array of slots. The first five slots are labeled with blue circles containing the numbers 5, 4, 3, 2, and 1 from top to bottom. Red dotted arrows indicate a shift of elements: the element 5 moves to the slot below 4, 4 moves to the slot below 3, 3 moves to the slot below 2, 2 moves to the slot below 1, and 1 moves to the slot below [pos]. The slots are labeled on the right as [0], [1], [2], ..., [pos], and [currentSize-1].

```
if (currentSize < values.length)
{
    currentSize++;
    for (int i = currentSize - 1; i > pos; i--)
    {
        values[i] = values[i - 1]; // move
        down
    }
    values[pos] = newElement; // fill
    hole
}
```

# Common Algorithms 8:

- Swapping Elements
  - Three steps using a temporary variable

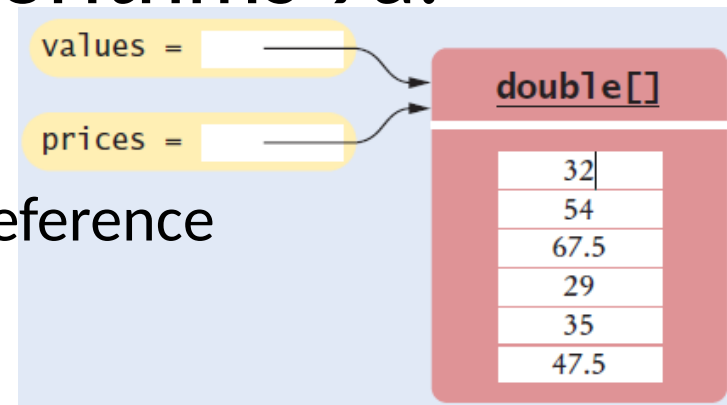


# Common Algorithms 9a:

- Copying Arrays

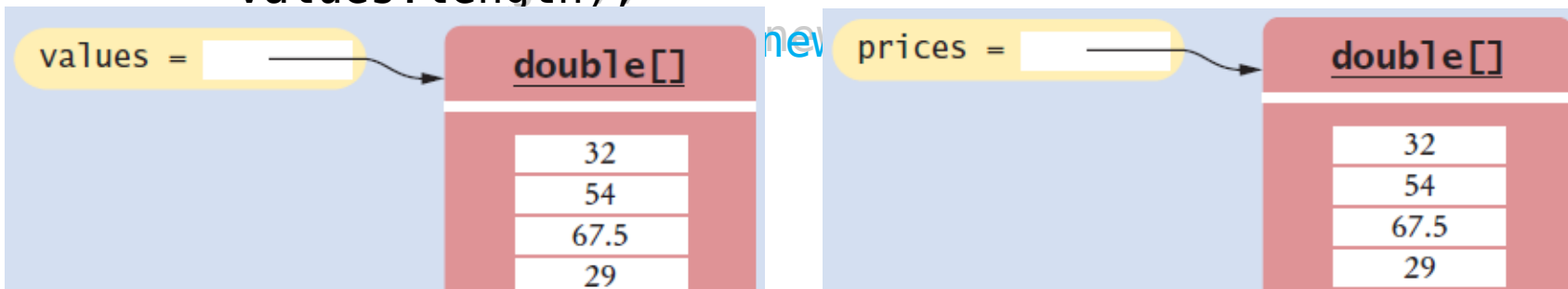
- Not the same as copying only the reference

- Copying creates two set of contents!



- Use the new (Java 6) `Arrays.copyOf` method

```
double[] values = new double[6];  
... // Fill array  
double[] prices = values; // Only a reference so far  
double[] prices = Arrays.copyOf(values,  
    values.length);
```





# Common Algorithms 9b:

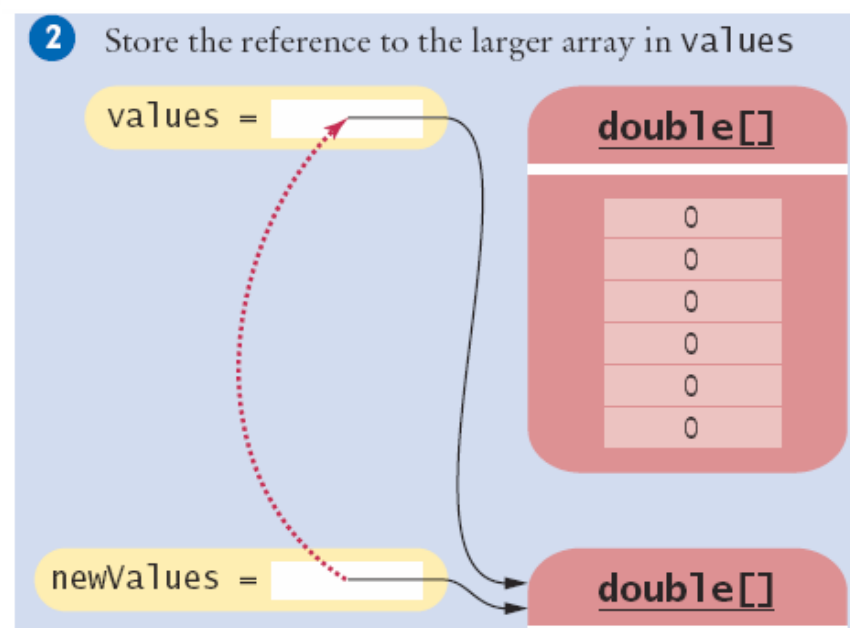
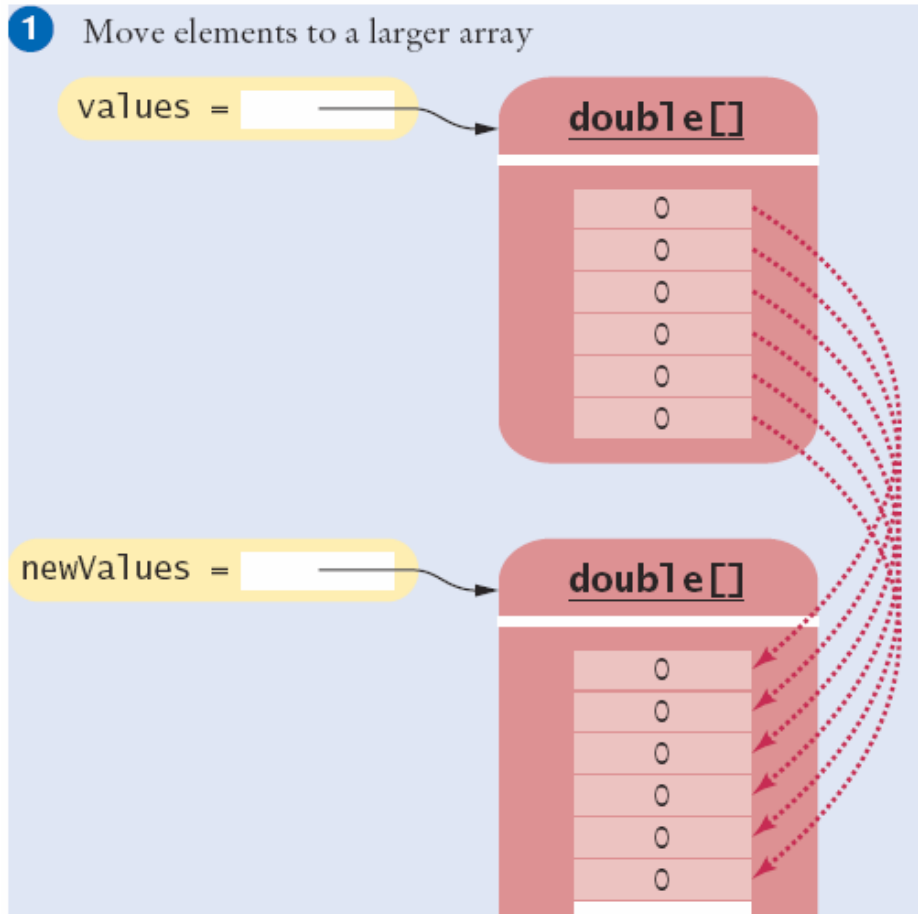
- Growing an array
  - Copy the contents of one array to a larger one
  - Change the reference of the original to the larger one
- Example: Double the size of an existing array
  - Use the `Arrays.copyOf` method
  - Use '`2 *`' in the second parameter

```
double[] values = new double[6];  
. . . // Fill array  
double[] newValues = Arrays.copyOf(values, 2 *  
    values.length);  
values = newValues;
```

`Arrays.copyOf` second parameter  
is the length of the new array

# Increasing the Size of an Array

- Copy all elements of values to newValues



- Then copy `newValues` reference over `values` reference

# Common Algorithms 10:

- Reading Input

- A: Known number of values to expect
  - Make an array that size and fill it one-by-one

```
double[] inputs = new double[NUMBER_OF_INPUTS];
for (i = 0; i < values.length; i++)
{
    inputs[i] = in.nextDouble();
}
```

```
double[] inputs = new double[MAX_INPUTS];
int currentSize = 0;
while (in.hasNextDouble() && currentSize < inputs.length)
{
    inputs[currentSize] = in.nextDouble();
    currentSize++;
}
```

# LargestInArray.java (1)

```
1  import java.util.Scanner;
2
3  /**
4   This program reads a sequence of values and prints them, marking the largest value.
5   */
6  public class LargestInArray
7  {
8      public static void main(String[] args)
9      {
10         final int LENGTH = 100;
11         double[] data = new double[LENGTH];
12         int currentSize = 0;
13
14         // Read inputs
15
16         System.out.println("Please enter values, Q to quit:");
17         Scanner in = new Scanner(System.in);
18         while (in.hasNextDouble() && currentSize < data.length)
19         {
20             data[currentSize] = in.nextDouble();
21             currentSize++;
22         }
```

Input values and store in next available index of the array

# LargestInArray.java (2)

```
24 // Find the largest value
25
26 double largest = data[0];
27 for (int i = 1; i < currentSize; i++)
28 {
29     if (data[i] > largest)
30     {
31         largest = data[i];
32     }
33 }
34
35 // Print all values, marking the largest
36
37 for (int i = 0; i < currentSize; i++)
38 {
39     System.out.print(data[i]);
40     if (data[i] == largest)
41     {
42         System.out.print(" <== largest value");
43     }
44     System.out.println();
45 }
46 }
47 }
```

Use a for loop and the 'Find the largest' algorithm

## Program Run

```
Please enter values, Q to quit:
35 80 115 44.5 Q
35
80
115 <== largest value
44.5
```

## Common Error 6.3



- Underestimating the Size of the Data Set
  - The programmer cannot know how someone might want to use a program!
  - Make sure that you write code that will politely reject excess input if you used fixed size limits

**Sorry, the number of lines of text is higher than expected, and some could not be processed. Please break your input into smaller size segments (1000 lines maximum) and run the program again.**

## 6.4 Using Arrays with Methods

- Methods can be declared to receive references as parameter variables
- What if we wanted to write a method to sum all of the elements in an array?

– Pass the array *reference* as an argument!

```
priceTotal =  
    sum(prices); reference
```

```
public static double sum(double[]  
    values)  
{  
    double total = 0;  
    for (double element : values)  
        total = total + element;  
    return total;  
}
```

prices =

double[]

32
54
67.5
29
35
47.5

Arrays can be used as  
method arguments and  
method return values.

# Passing References

- Passing a reference give the called method access to all of the data elements
  - It CAN change the values!
- Example: Multiply each element in the passed array by the value passed in the second parameter
  - The parameter variables `values` and `factor` are created. 1

```
multiply(values, 10);
```

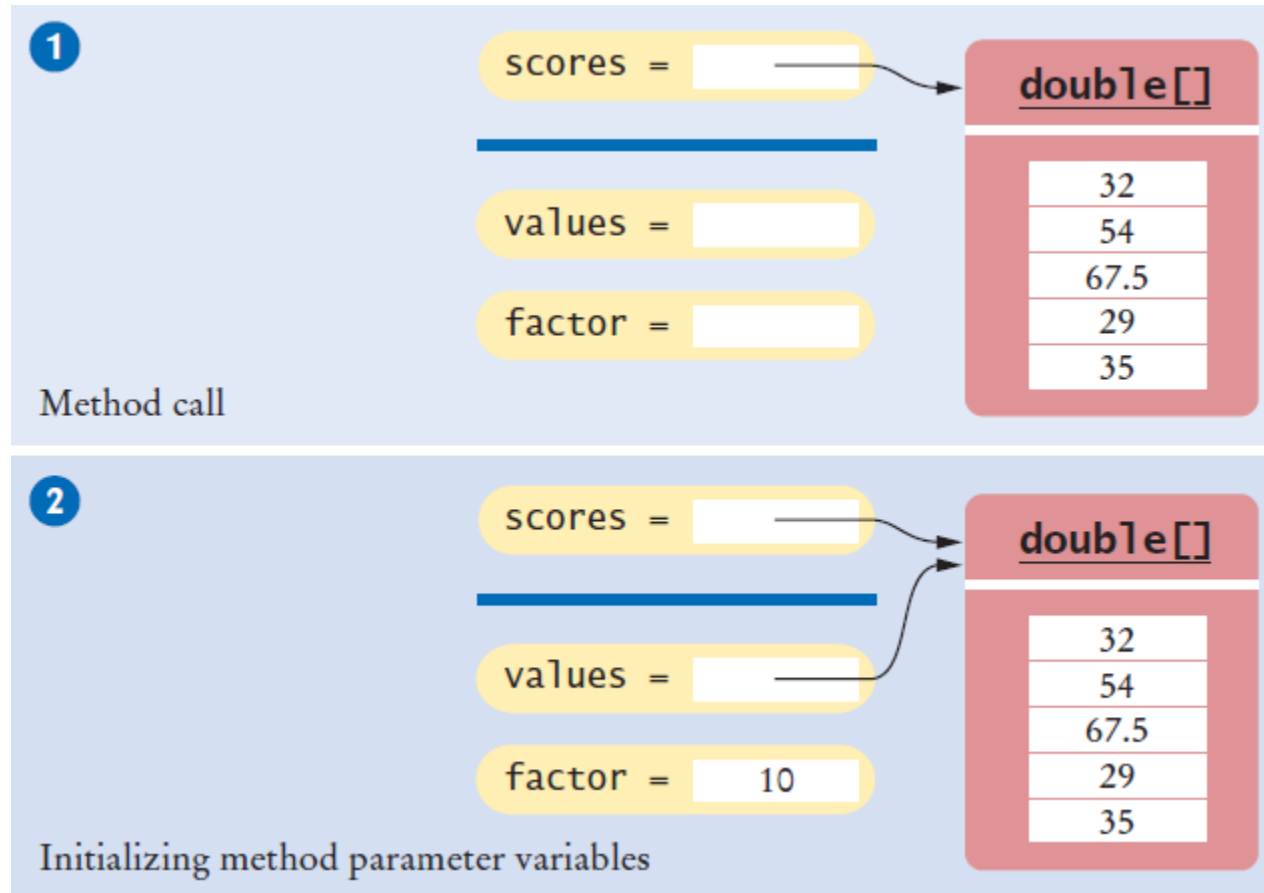
reference

value

```
public static void multiply(double[] data, double
factor)
{
    for (int i = 0; i < data.length; i++)
        data[i] = data[i] * factor;
}
```

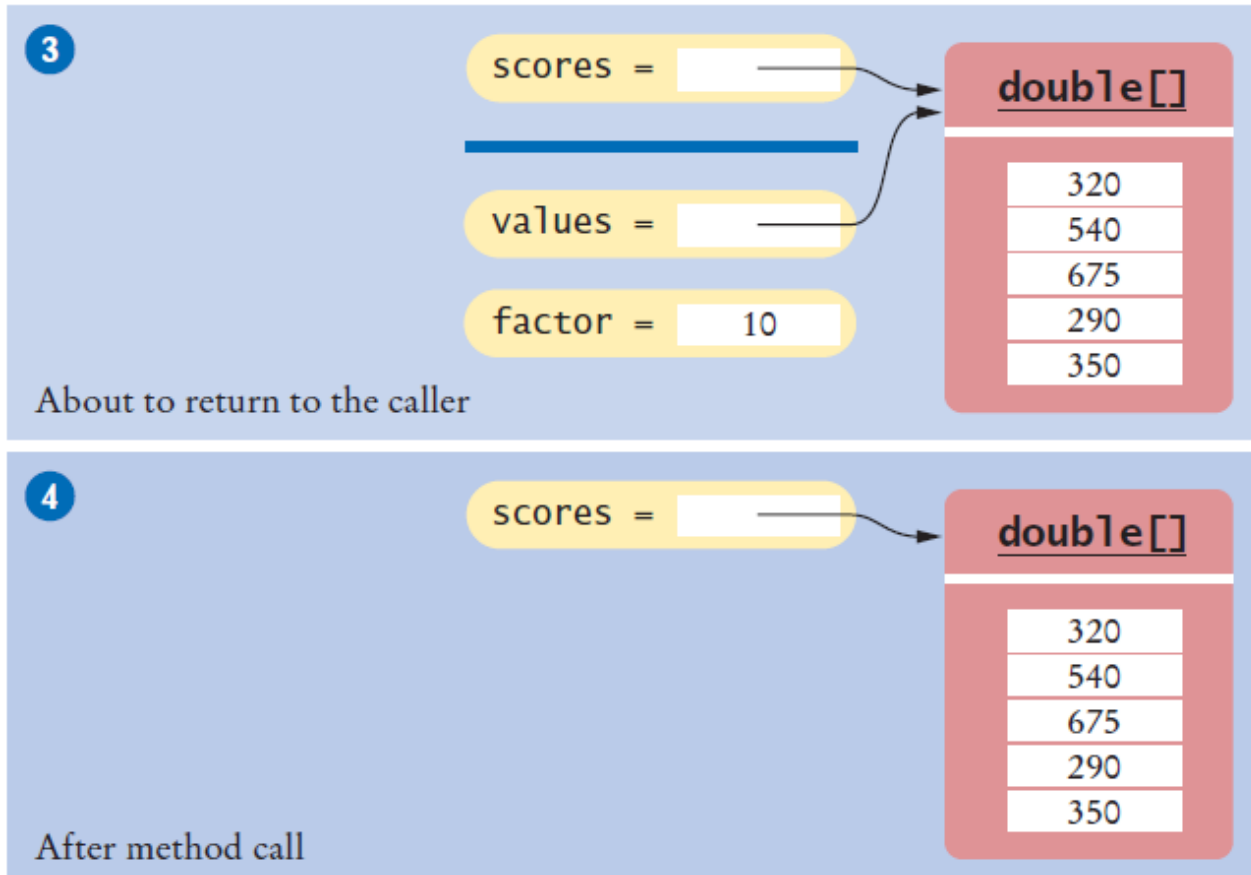


# Passing References (Step 2)



- The parameter variables are initialized with the arguments that are passed in the call. In our case, values is set to scores and factor is set to 10. Note that values and scores are references to the *same* array. **2**

# Passing References (Steps 3 & 4)



- The method multiplies all array elements by 10. **3**
- The method returns. Its parameter variables are removed. However, values still refers to the array with the modified values. **4**

# Method Returning an Array

- Methods can be declared to return an array

```
public static int[] squares(int  
n)
```

- To Call: Create a compatible array reference:

```
int[] numbers = squares(10);
```

value

– Call the method

r  
e  
f  
e  
r  
e  
n  
c  
e

```
public static int[] squares(int n)  
{  
    int[] result = new int[n];  
    for (int i = 0; i < n; i++)  
    {  
        result[i] = i * i;  
    }  
    return result;  
}
```

## 6.5 Problem Solving

- Adapting Algorithms
- Consider this example problem: You are given the quiz scores of a student. You are to compute the final quiz score, which is the sum of all scores after dropping the lowest one.
  - For example, if the scores are  
8   7   8.5   9.5   7   5   10
  - then the final score is 50.

# Adapting a Solution

- What steps will we need?
  - Find the minimum.
  - Remove it from the array.
  - Calculate the sum.
- What tools do we know?
  - Finding the minimum value (Section 6.3.3)
  - Removing an element (Section 6.3.6)
  - Calculating the sum (Section 6.3.2)
- But wait... We need to find the POSITION of the minimum value, not the value itself..
  - Hmmm. Time to adapt

[0]	[1]	[2]	[3]	[4]	[5]	[6]
8	7	8.5	9.5	7	4	10

# Planning a Solution

- Refined Steps:
  - Find the minimum value.
  - Find it's position.
  - Remove it from the array.
  - Calculate the sum.

- Let's try it

- Find the position of the minimum:

- At position 5

[0]	[1]	[2]	[3]	[4]	[5]	[6]
8	7	8.5	9.5	7	4	10

[0]	[1]	[2]	[3]	[4]	[5]	[6]
8	7	8.5	9.5	7	4	10

- Remove it from the array
  - Calculate the sum

[0]	[1]	[2]	[3]	[4]	[5]
8	7	8.5	9.5	7	10

# Adapting the code

- Adapt smallest value to smallest position:

```
double smallest = values[0];
for (int i = 1; i < values.length;
    i++)
{
    if (values[i] < smallest)
    {
        smallest = values[i];
    }
}

int smallestPosition = 0;
for (int i = 1; i < values.length; i++)
{
    if (values[i] <
        values[smallestPosition] )
    {
        smallestPosition = i;
    }
}
```

# Using Arrays with Methods

1) Decompose the task into steps

Read inputs.  
Remove the minimum.  
Calculate the sum.

2) Determine the algorithms to use

Read inputs.  
Find the minimum.  
Find its position.  
Remove the minimum.  
Calculate the sum.

3) Use methods to structure the program

```
double[] scores = readInputs();  
double total = sum(scores) -  
    minimum(scores);  
System.out.println("Final score: " +  
    total);
```

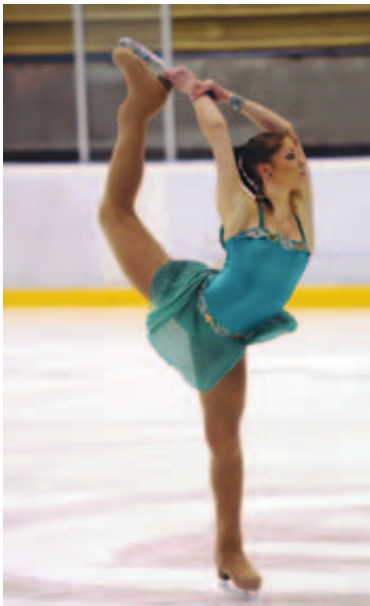
- readInputs
- sum
- minimum

4) Assemble and Test the program



## 6.7 Two-Dimensional Arrays

- Arrays can be used to store data in two dimensions (2D) like a spreadsheet
  - Rows and Columns
  - Also known as a 'matrix'



	Gold	Silver	Bronze
Canada	1	0	1
China	1	1	0
Germany	0	0	1
Korea	1	0	0
Japan	0	1	1
Russia	0	1	1
United States	1	1	0

**Figure 12** Figure Skating Medal Counts

# Declaring Two-Dimensional Arrays

- Use two 'pairs' of square braces

```
const int COUNTRIES = 7;  
const int MEDALS = 3;  
int[][] counts = new int[COUNTRIES]  
[MEDALS];
```

- You can also initialize the array

```
const int COUNTRIES =  
7;  
const int MEDALS = 3;  
int[][] counts =  
{  
    { 1, 0, 1 },  
    { 1, 1, 0 },  
    { 0, 0, 1 },  
    { 1, 0, 0 },  
    { 0, 1, 1 },  
    { 0, 1, 1 },  
    { 1, 1, 0 }  
};
```

Gold	Silver	Bronze
1	0	1
1	1	0
0	0	1
1	0	0
0	1	1
0	1	1
1	1	0

Note the use of two 'levels' of curly braces. Each row has braces with commas separating them.

## Syntax 6.3: 2D Array Declaration

Diagram illustrating the syntax for a 2D array declaration:

```
double[][] tableEntries = new double[7][3];
```

Labels:

- Name: `tableEntries`
- Element type: `double`
- Number of rows: `7`
- Number of columns: `3`

All values are initialized with 0.

Diagram illustrating the syntax for a 2D array declaration with initial values:

```
int[][] data = {  
    { 16, 3, 2, 13 },  
    { 5, 10, 11, 8 },  
    { 9, 6, 7, 12 },  
    { 4, 15, 14, 1 },  
};
```

Labels:

- Name: `data`
- List of initial values: The inner arrays.

- The name of the array continues to be a reference to the contents of the array
  - Use `new` or fully initialize the array

# Accessing Elements

- Use two index values:

Row then Column

```
int value = counts[3]
```

- To print  
[1];

- Use nested for loops

- Outer row(i) inner column(i).

```
for (int i = 0; i < COUNTRIES; i++)  
{
```

```
    // Process the ith row
```

```
    for (int j = 0; j < MEDALS; j++)
```

```
    {
```

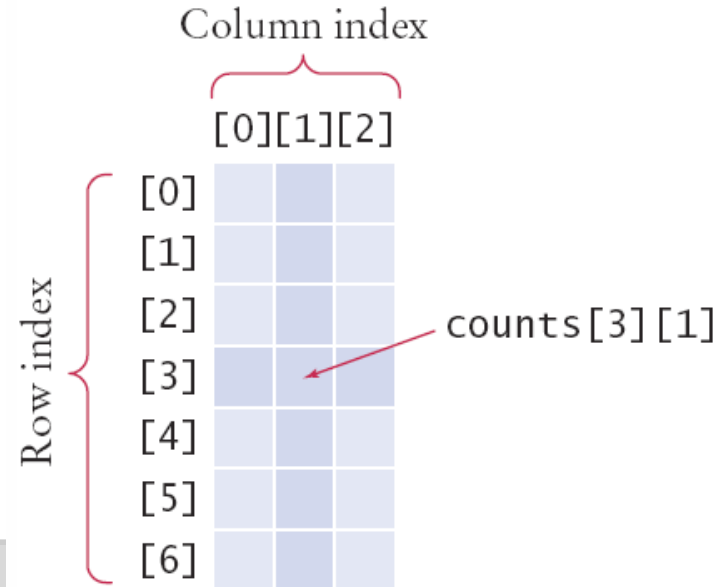
```
        // Process the jth column in the ith row
```

```
        System.out.printf("%8d", counts[i][j]);
```

```
    }
```

```
    System.out.println(); // Start a new line at the end of  
    the row
```

```
}
```



# Locating Neighboring Elements

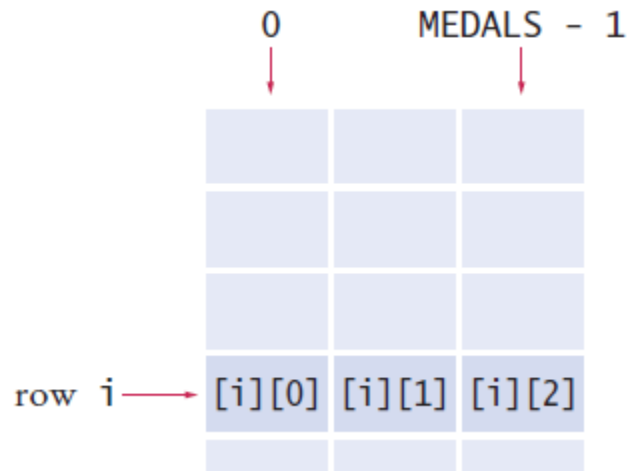
- Some programs that work with two-dimensional arrays need to locate the elements that are adjacent to an element
- This task is particularly common in games
- You are at loc  $i, j$
- Watch out for edges!
  - No negative indexes!
  - Not off the 'board'

$[i - 1][j - 1]$	$[i - 1][j]$	$[i - 1][j + 1]$
$[i][j - 1]$	$[i][j]$	$[i][j + 1]$
$[i + 1][j - 1]$	$[i + 1][j]$	$[i + 1][j + 1]$

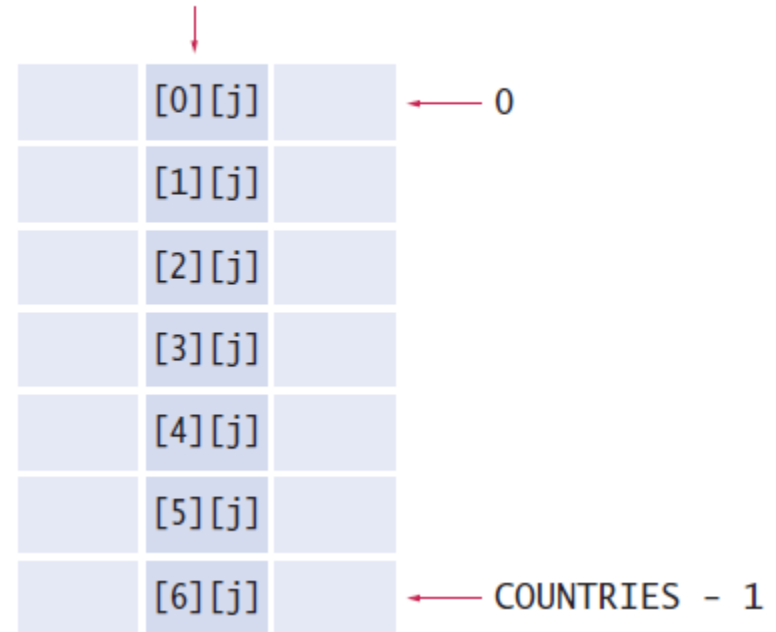
# Adding Rows and Columns

- Rows (x)

```
int total = 0;
for (int j = 0; j < MEDALS;
    j++)
{
    total = total + counts[i]
    [i]:
}
```



- Columns (y)



```
int total = 0;
for (int i = 0; i < COUNTRIES;
    i++)
{
    total = total + counts[i][j];
}
```