

CHAPTER

6

# ARRAYS and ARRAYLISTS





# 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!

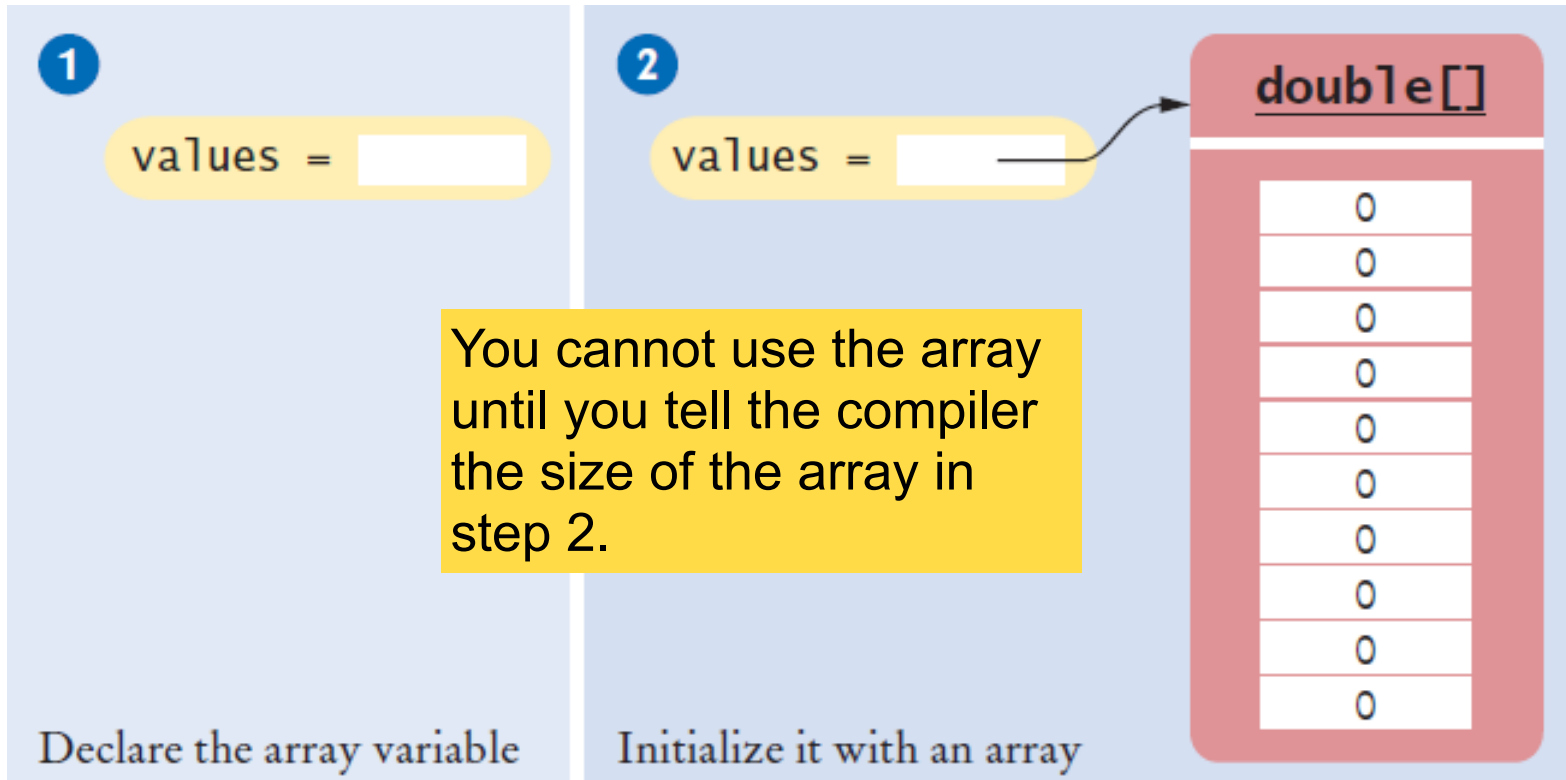
32
54
67.5
29
35
80
115
44.5
100
65



# Declaring an Array

❑ Declaring an array is a two step process

- 1) `double[] values;`    *// declare array variable*
- 2) `values = new double[10];`    *// initialize array*





# Declaring an Array (Step 1)

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

Type  
`double`

Square Braces  
`[ ]`

Array name  
`values`

semicolon  
`;`

- 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

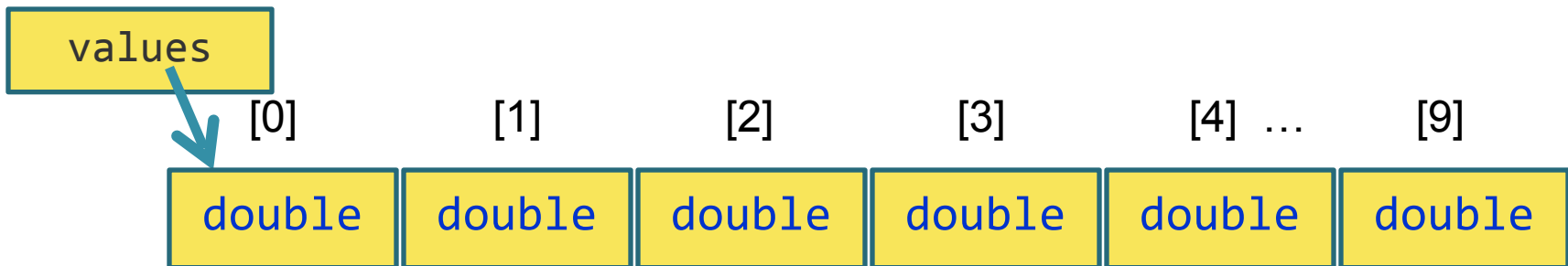


# Declaring an Array (Step 2)

- ❑ Reserve memory for all of the elements:

Array name	Keyword	Type	Size	semicolon	
<code>values</code>	<code>=</code>	<code>new</code>	<code>double</code>	<code>[10]</code>	<code>;</code>

- 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
- I don't like this: makes it look like type is double.
  - It's NOT





## Couple of Notes (cont.)

- ⌚ First: an array is a type
  - ⌚ an **object** type
  - ⌚ Ex. `int[]` is a type: an array of ints (or simply an int array)
  - ⌚ Ex. `String[]` is a type: an array of Strings (or simply a String array)
- ⌚ Since an array is an object type, what gets stored in a variable of any of the array types is an object reference (even if the base type of the array is a primitive type!)



# Important

- ⌚ To be perfectly clear, some terminology
- ⌚ The **base type** of an array is the type that each individual array element has
  - ⌚ E.g., `int[] pixel;`
  - ⌚ `pixel` is of type int array, but the base type of `pixel` is `int` (because all elements of the array are ints)
- ⌚ If the base type of your array is an object type, then you **must** initialize (allocate memory for) each element of the array before using that element!



# Couple of Notes

- ⌚ Second, note the AMAZING ability an array gives you: you can declare a huge number of variables with a single line of code!
  - ⌚ `int[] myArray = new int[1000000];`
  - ⌚ If you think you won't need that many variables in a program, see Labs 6,7,8, and final project!



# 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
  - **Truth be told, this is rarely used**



# 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;
}
```

3

values =

double[]

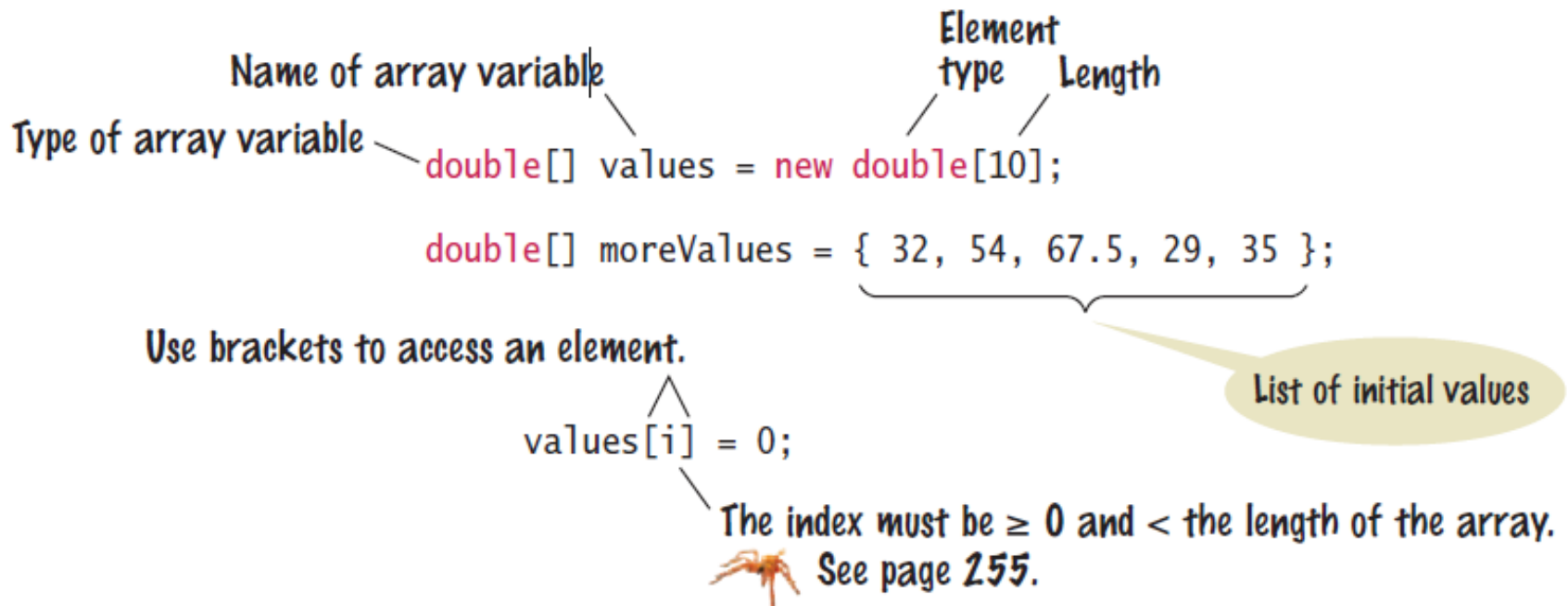
[0]	0
[1]	0
[2]	0
[3]	0
[4]	35
[5]	0
[6]	0
[7]	0
[8]	0
[9]	0

Access an array element



# Syntax 6.1: Array

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





# Array Index Numbers

- ❑ Array index numbers start at 0
  - The rest are positive integers
- ❑ An array with 10 element has indexes 0 - 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 <code>double[]</code> variable with an array of type <code>int[]</code> .



# 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

int[]

10

9

7

4

5

Values

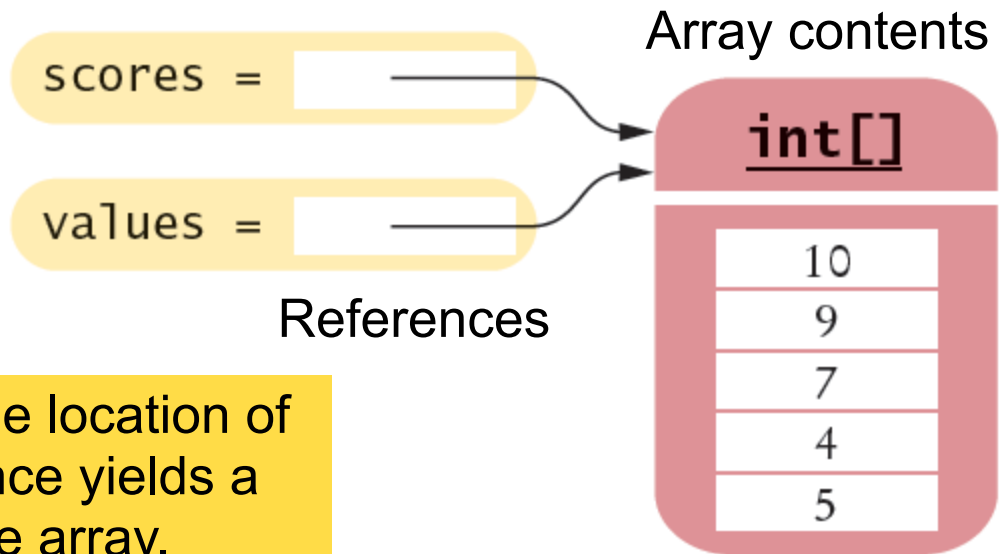
An array variable contains a *reference* to the array contents. The *reference* is the location of the array contents (in memory).



# 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)

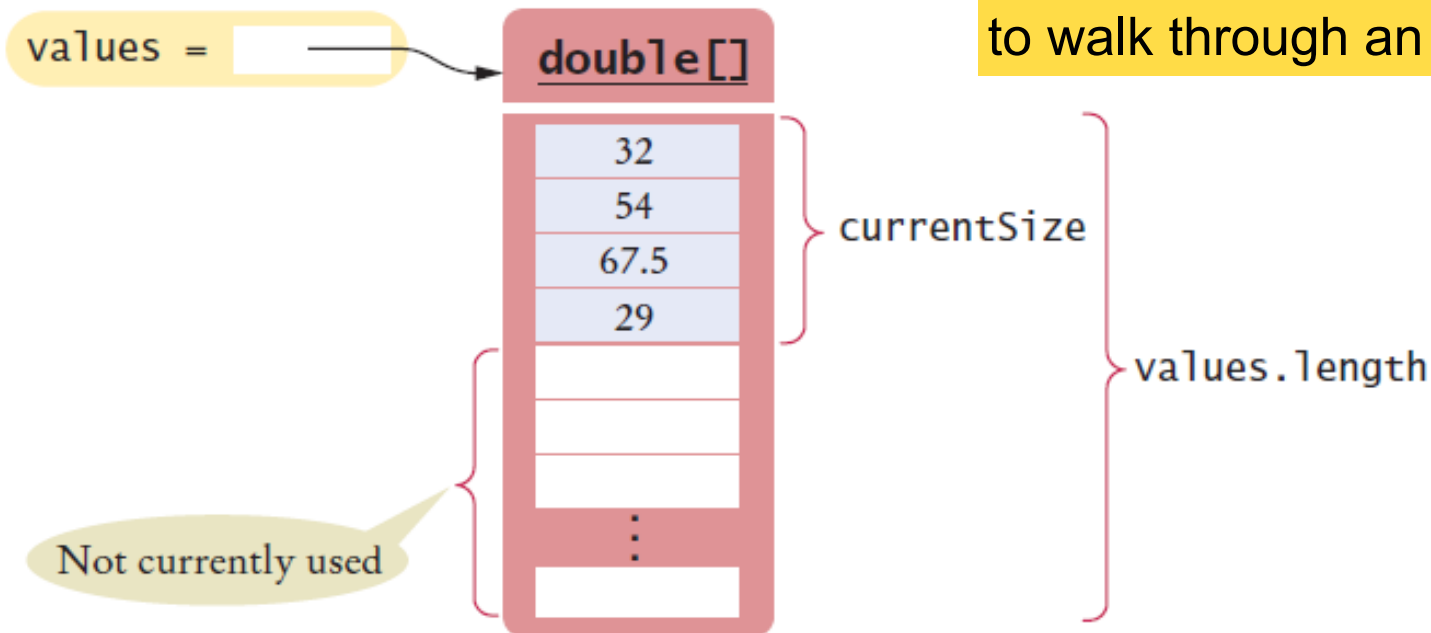


# Walking a Partially Filled Array

- 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
```

1

values =

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

```
double[] values;  
values = new double[10];  
values[0] = 29.95; // No error
```

2

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



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

[32, 54, 67.5, 29, 35]



# 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++;
    }
}
```

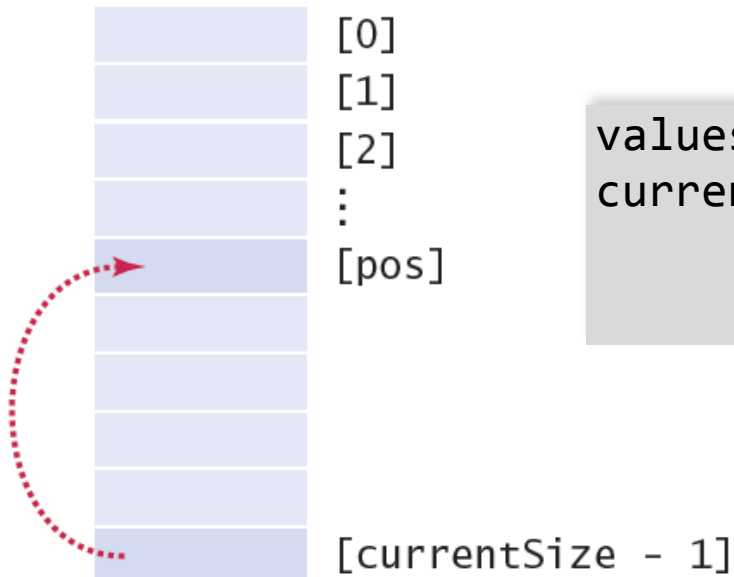
Compound condition to prevent bounds error if value not found.

```
if (found)
{
    System.out.println("Found at position: " + pos);
}
else { System.out.println("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

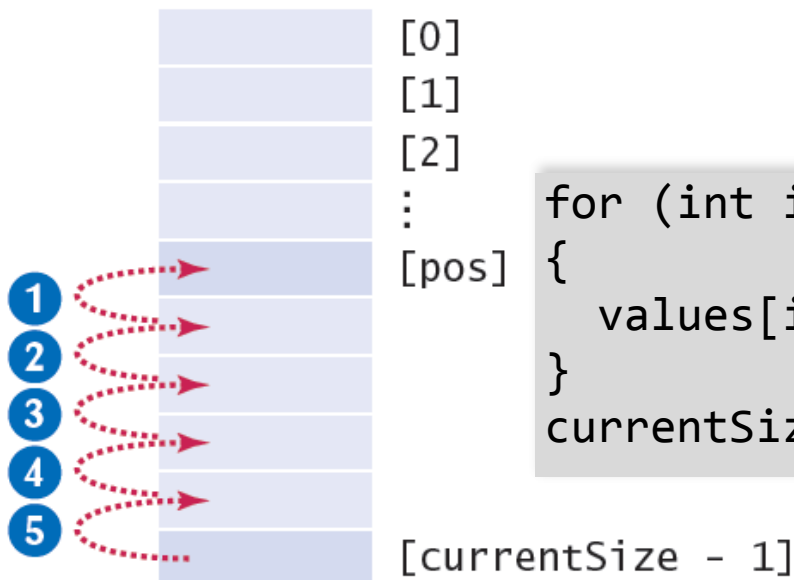


```
values[pos] = values[currentSize - 1];  
currentSize--;
```



# 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



```
for (int i = pos; i < currentSize - 1; i++)  
{  
    values[i] = values[i + 1];  
}  
currentSize--;
```

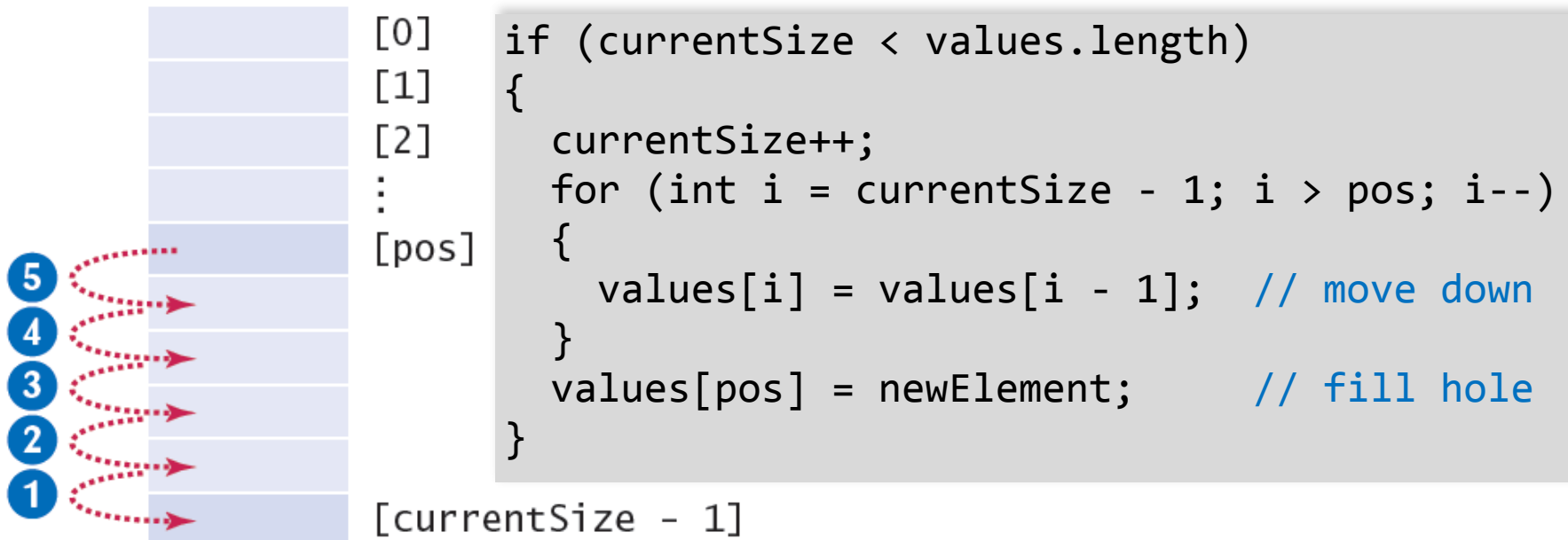




# Common Algorithms 7:

## ❑ 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

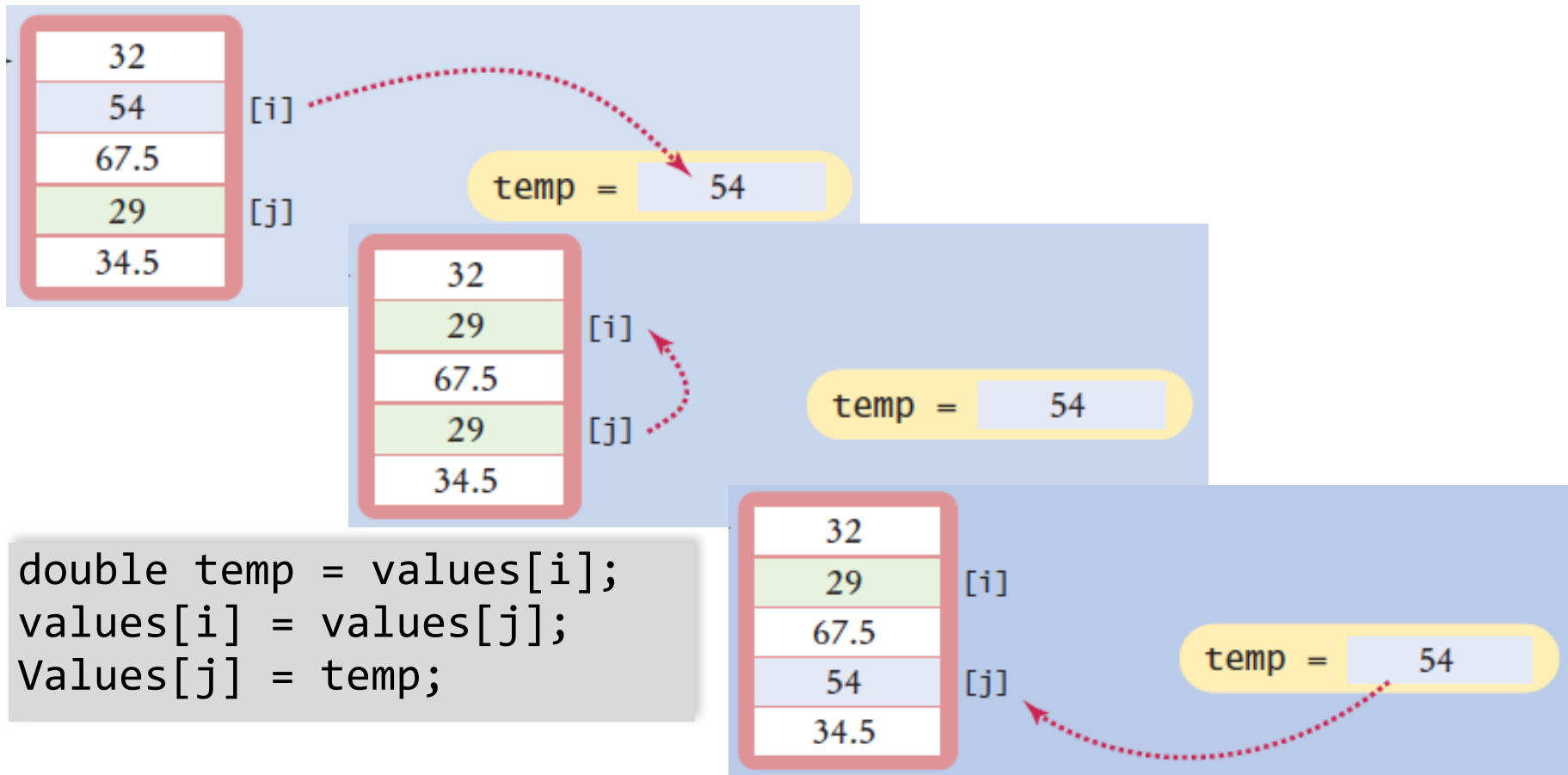




# Common Algorithms 8:

## ❑ Swapping Elements

- Three steps using a temporary variable

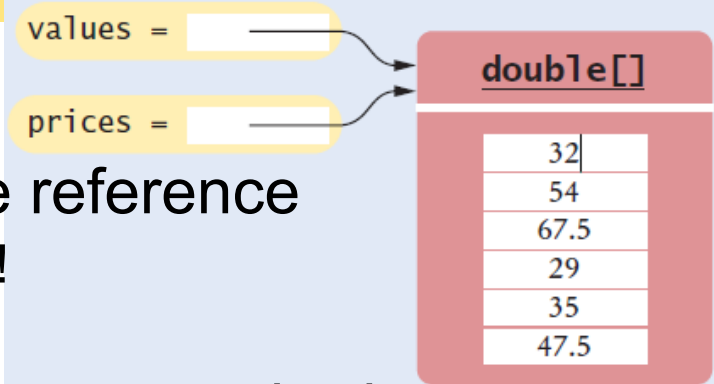




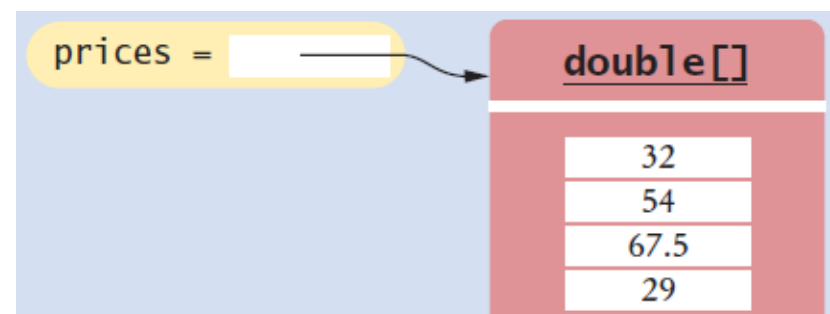
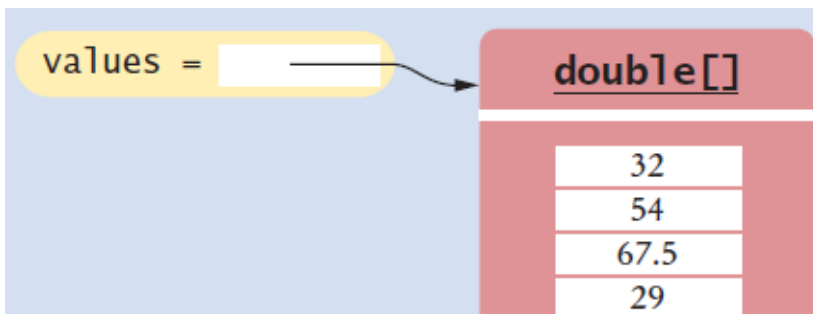
# Common Algorithms 9a:

## ❑ 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);  
// copyOf creates the new copy, returns a reference
```





## 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;
```

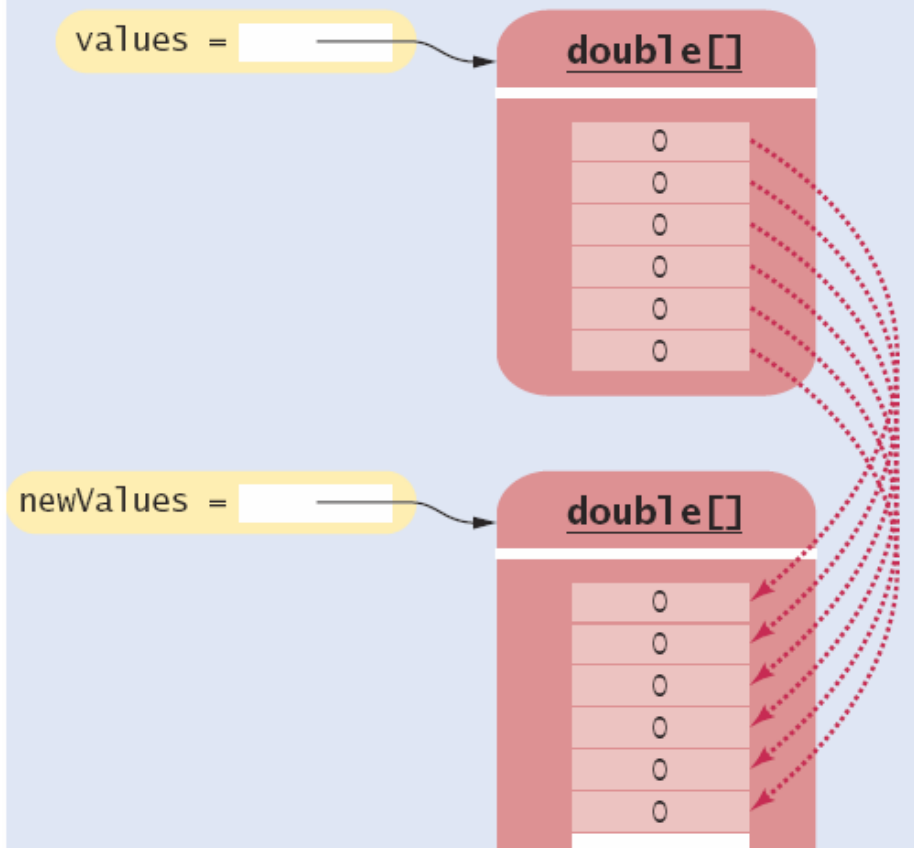
`Arrays.copyOf` second parameter is the length of the new array



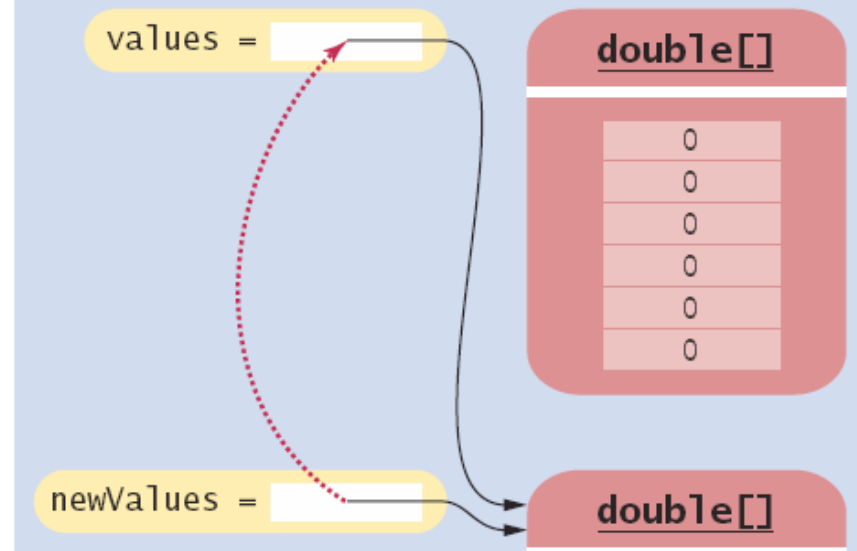
# Increasing the Size of an Array

- Copy all elements of values to newValues

1 Move elements to a larger array



2 Store the reference to the larger array in values



- 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();
}
```

- B: Unknown number of values
  - Make maximum sized array, maintain as partially filled array

```
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.`



# Special Topic: Sorting Arrays

- ❑ When you store values into an array, you can choose to either:

- Keep them unsorted (random order)

[0]	[1]	[2]	[3]	[4]
11	9	17	5	12

- Sort them (Ascending or Descending...)

[0]	[1]	[2]	[3]	[4]
5	9	11	12	17

- ❑ A sorted array makes it much easier to find a specific value in a large data set
- ❑ The Java API provides an efficient sort method:

```
Arrays.sort(values);           // Sort all of the array  
Arrays.sort(values, 0, currentSize); // partially filled
```



# Special Topic: Searching

- ❑ We have seen the Linear Search (6.3.5)
  - It works on an array that is sorted, or unsorted
  - Visits each element (start to end), and stop if you find a match or find the end of the array
- ❑ Binary Search
  - Only works for a sorted array
  - Compare the middle element to our target
    - If it is lower, exclude the lower half
    - If it is higher, exclude the higher half
  - Do it again until you find the target or you cannot split what is left



# Binary Search

## ❑ Binary Search

- Only works for a sorted array
- Compare the middle element to our target
  - If it is lower, exclude the lower half
  - If it is higher, exclude the higher half
  - Do it again until you find the target or you cannot split what is left
- Example: Find the value 15

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
1	5	8	9	12	17	20	32

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
1	5	8	9	12	17	20	32

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
1	5	8	9	12	17	20	32

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
1	5	8	9	12	17	20	32

**Sorry, 15 is not in this array.**



# Binary Search Example

```
boolean found = false, int low = 0, int pos = 0;
int high = values.length - 1;

while (low <= high && !found)
{
    pos = (low + high) / 2; // Midpoint of the subsequence
    if (values[pos] == searchedValue)
        { found = true; } // Found it!
    else if (values[pos] < searchedValue)
        { low = pos + 1; } // Look in first half
    else { high = pos - 1; } // Look in second half
}
if (found)
    { System.out.println("Found at position " + pos); }
else
    { System.out.println("Not found. Insert before position " + pos); }
```

[0][1][2][3][4][5][6][7]  
1 5 8 9 12 17 20 32

[0][1][2][3][4][5][6][7]  
1 5 8 9 12 17 20 32

[0][1][2][3][4][5][6][7]  
1 5 8 9 12 17 20 32



## 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);
```

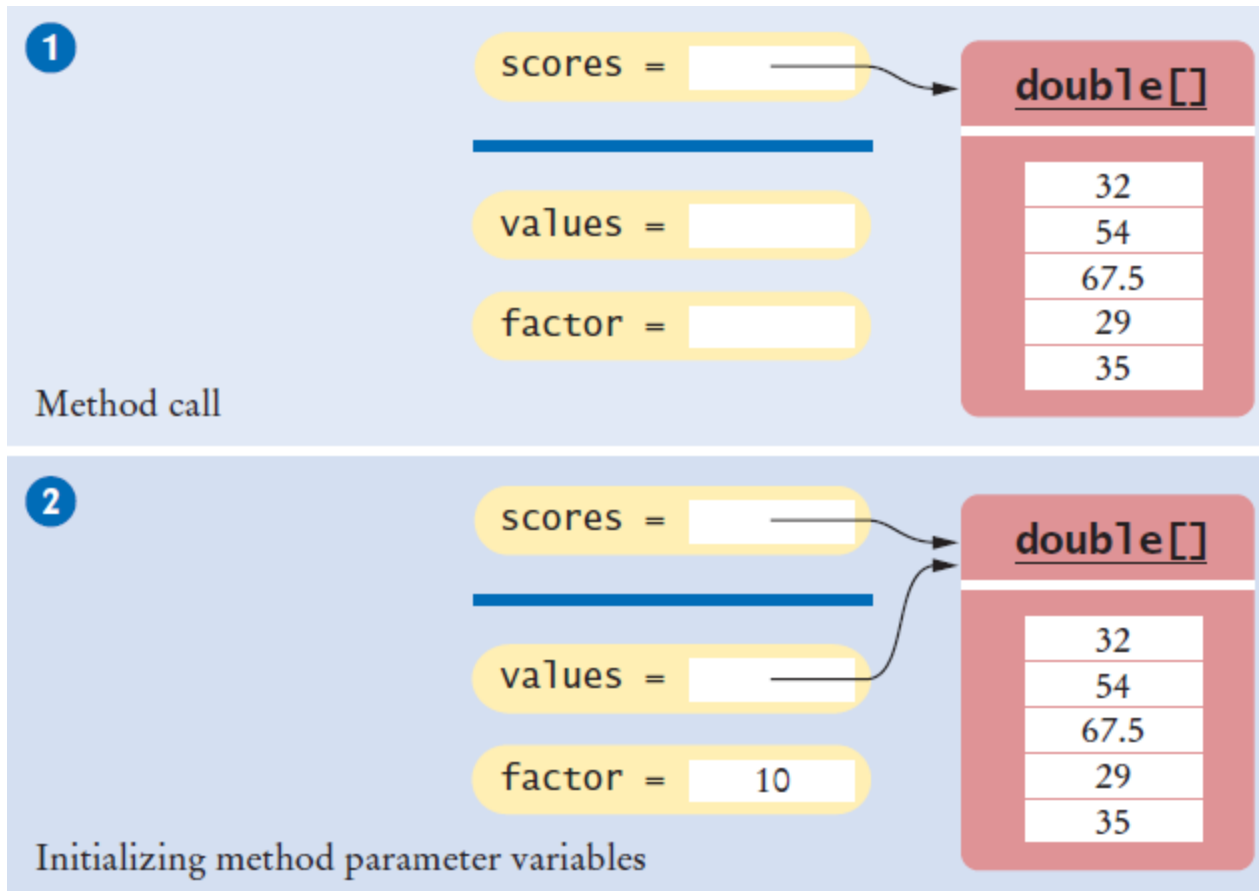
reference

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)

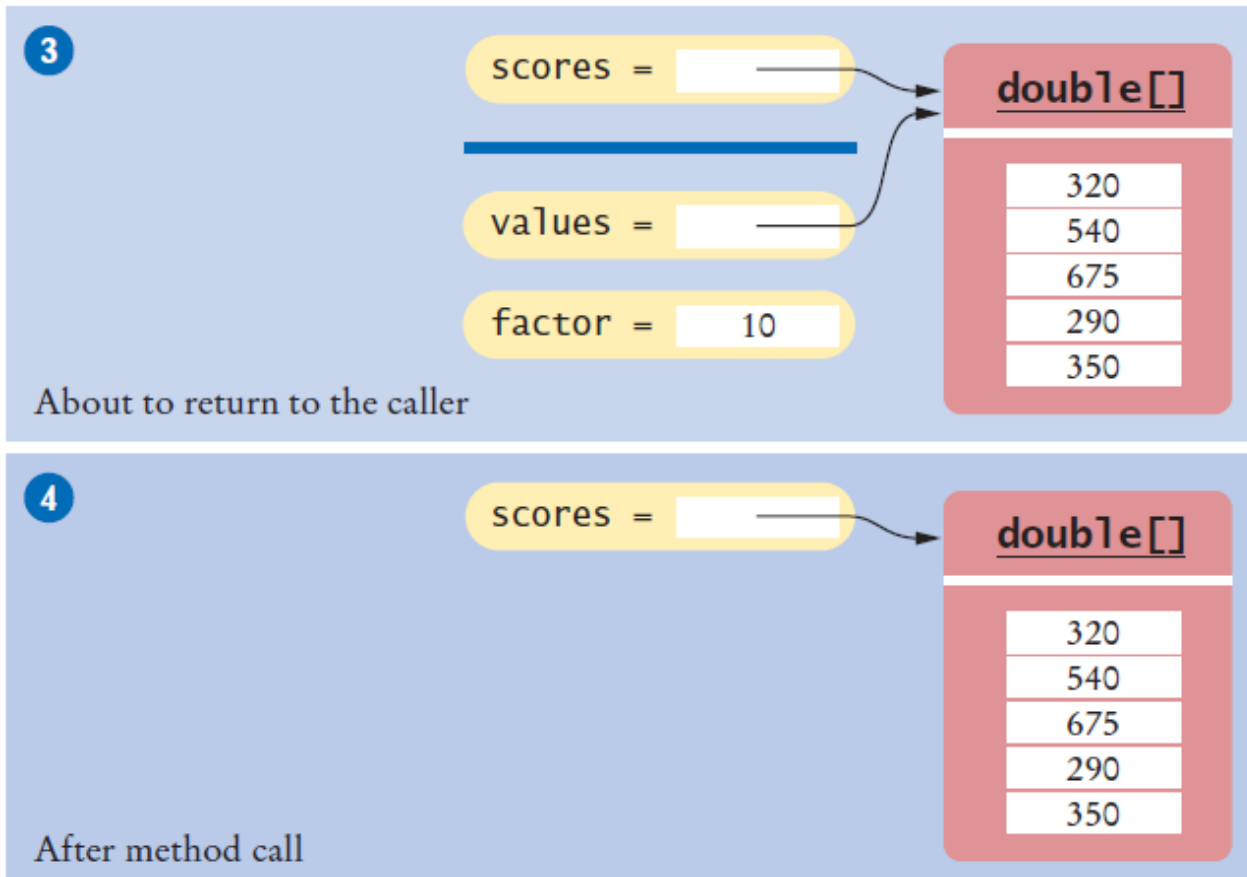


- The parameter variables are initialized with the arguments that are passed in the call. In our case, `values` is set to `scores` and `factor` is set to 10. Note that `values` and `scores` are references to the *same* array. **2**





# 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);
```

- Call the method

value

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

reference



# 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



# Assembling and Testing

- ❑ Place methods into a class
- ❑ Review your code
  - Handle exceptional situations?
    - Empty array?
    - Single element array?
    - No match?
    - Multiple matches?

Test Case	Expected Output	Comment
8 7 8.5 9.5 7 5 10	50	See Step 1.
8 7 7 9	24	Only one instance of the low score should be removed.
8	0	After removing the low score, no score remains.
(no inputs)	<b>Error</b>	That is not a legal input.



## 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];
```

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

- ❑ 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 }  
};
```

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 pointing to the code:

- 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 pointing to the code:

- Name: `data`

List of initial values

- ❑ 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:

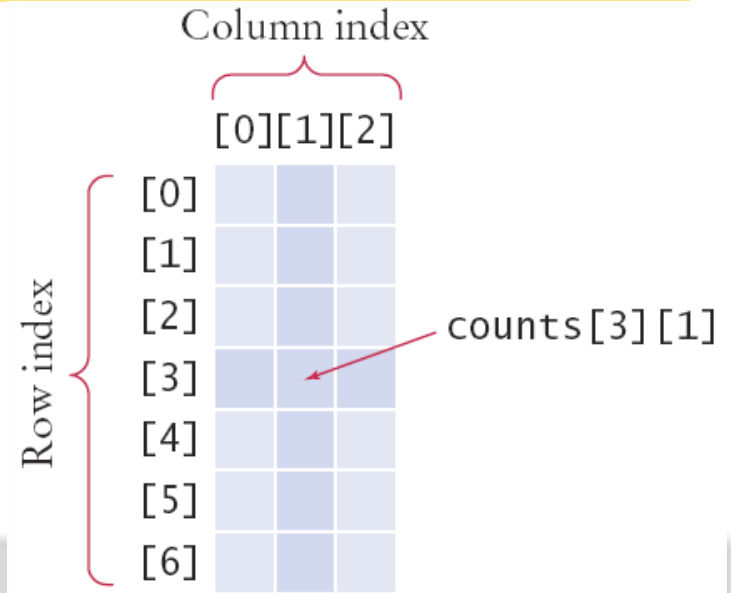
Row then Column

```
int value = counts[3][1];
```

- To print

- Use nested for loops
- Outer row(*i*) , inner column(*j*) :

```
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'

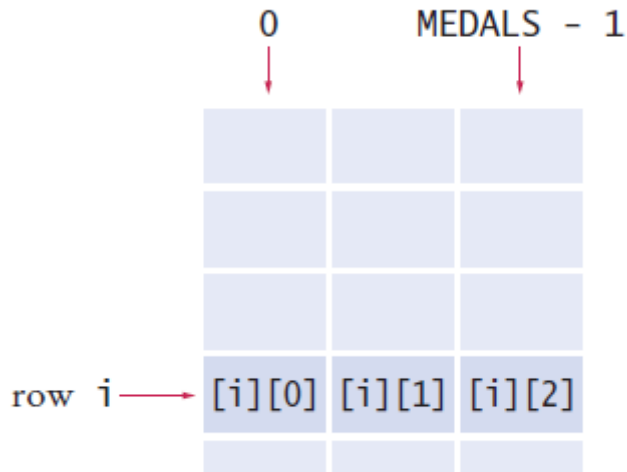
$[i - 1][j - 1]$	$[i - 1][j]$	$[i - 1][j + 1]$
$[i][j - 1]$	$[i][j]$	$[i][j + 1]$
$[i + 1][j - 1]$	$[i + 1][j]$	$[i + 1][j + 1]$



# Adding Rows and Columns

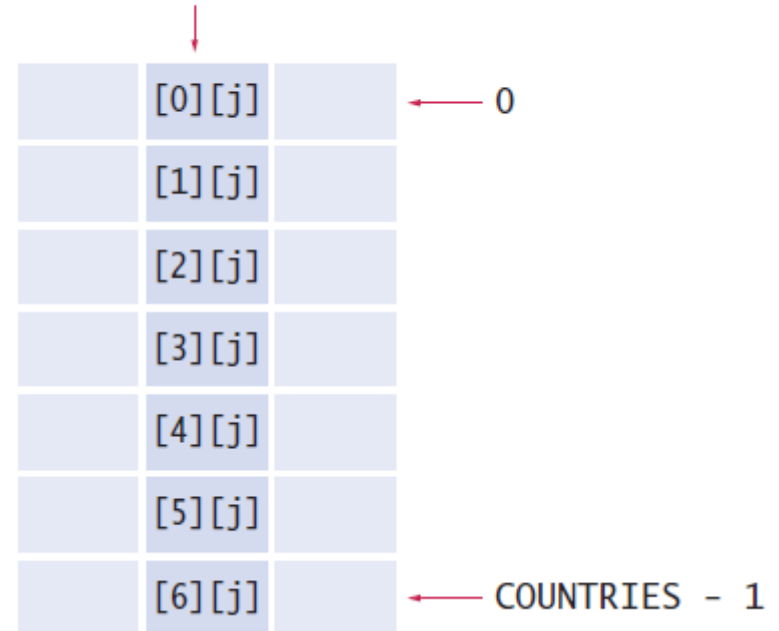
## □ Rows (x)

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



## Columns (y)

column j



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



# Medals.java (1)

```
1  /**
2   * This program prints a table of medal winner counts with row totals.
3   */
4  public class Medals
5  {
6      public static void main(String[] args)
7      {
8          final int COUNTRIES = 7;
9          final int MEDALS = 3;
10
11         String[] countries =
12             {
13                 "Canada",
14                 "China",
15                 "Germany",
16                 "Korea",
17                 "Japan",
18                 "Russia",
19                 "United States"
20             };
21
22         int[][] counts =
23             {
24                 { 1, 0, 1 },
25                 { 1, 1, 0 },
26                 { 0, 0, 1 },
27                 { 1, 0, 0 },
28                 { 0, 1, 1 },
29                 { 0, 1, 1 },
30                 { 1, 1, 0 }
```



# Medals.java (2)

```
33      System.out.println("          Country    Gold  Silver  Bronze  Total");
34
35      // Print countries, counts, and row totals
36      for (int i = 0; i < COUNTRIES; i++)
37      {
38          // Process the ith row
39          System.out.printf("%15s", countries[i]);
40
41          int total = 0;
42
43          // Print each row element and update the row total
44          for (int j = 0; j < MEDALS; j++)
45          {
46              System.out.printf("%8d", counts[i][j]);
47              total = total + counts[i][j];
48          }
49
50          // Display the row total and print a row separator
51          System.out.printf("%8d\n", total);
52      }
53  }
54 }
```

## Program Run

Country	Gold	Silver	Bronze	Total
Canada	1	0	1	2
China	1	1	0	2
Germany	0	0	1	1
Korea	1	0	0	1
Japan	0	1	1	2
Russia	0	1	1	2
United States	1	1	0	2



## 6.8 Array Lists

- ❑ When you write a program that collects values, you don't always know how many values you will have.
- ❑ In such a situation, a Java Array List offers two significant advantages:
  - Array Lists can grow and shrink as needed.
  - The ArrayList class supplies methods for common tasks, such as inserting and removing elements.

An Array List expands to hold as many elements as needed



# Declaring and Using Array Lists

- ❑ The ArrayList class is part of the `java.util` package
  - It is a *generic* class
    - Designed to hold many types of objects
  - Provide the type of element during declaration
    - Inside `< >` as the 'type parameter':
    - The type must be a Class
    - Cannot be used for primitive types (`int`, `double`...)

```
ArrayList<String> names = new ArrayList<String>();
```



## Syntax 6.4: Array Lists

Variable type      Variable name      An array list object of size 0

```
ArrayList<String> friends = new ArrayList<String>();
```

Use the  
get and set methods  
to access an element.

```
friends.add("Cindy");  
String name = friends.get(i);  
friends.set(i, "Harry");
```

The add method  
appends an element to the array list,  
increasing its size.

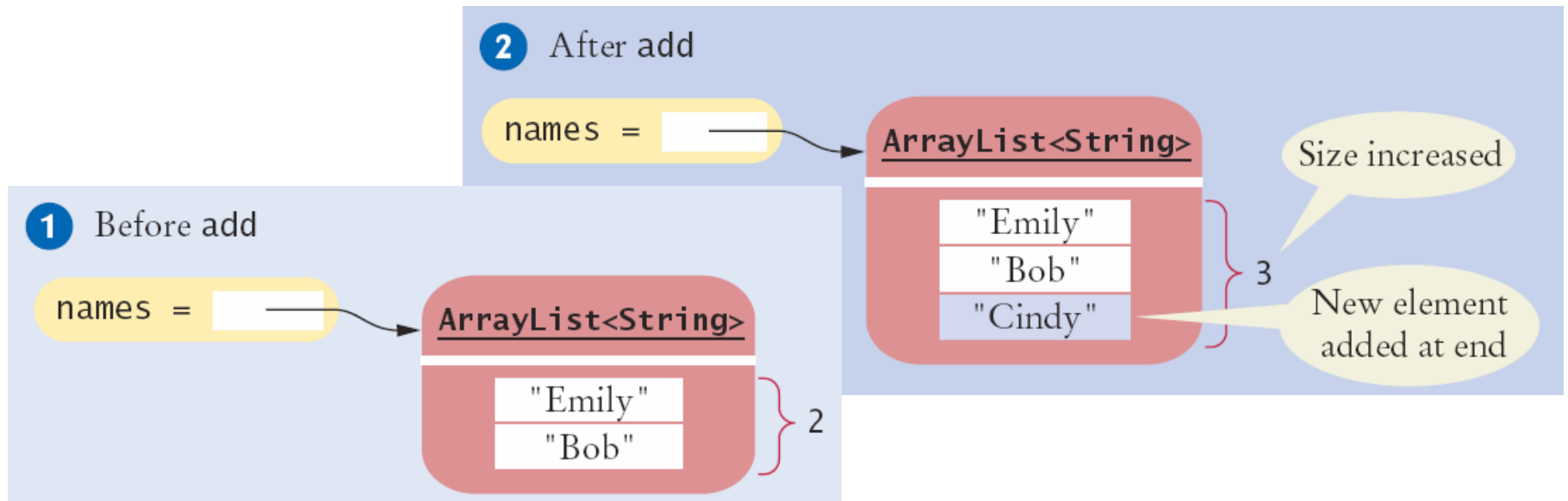
The index must be  
≥ 0 and < friends.size().

### ArrayList provides many useful methods:

- add: add an element
- get: return an element
- remove: delete an element
- set: change an element
- size: current length



# Adding an element with add()



❑ The **add** method has two versions:

- Pass a new element to add to the end  
`names.add("Cindy");`

- Pass a location (index) and the new value to add

`names.add(1, "Cindy");`

Moves all other elements





# Adding an Element in the Middle

1 Before add

names =

ArrayList<String>

"Emily"

"Bob"

"Carolyn"

```
names.add(1, "Ann");
```

ArrayList<String>

"Emily"

"Ann"

"Bob"

"Carolyn"

New element  
added at index 1

Moved from index 1 to 2

Moved from index 2 to 3

- Pass a location (index) and the new value to add  
Moves all other elements



# Removing an Element

names =

ArrayList<String>

"Emily"

"Ann"

"Bob"

"Carolyn"

```
names.remove(1);
```

ArrayList<String>

"Emily"

"Bob"

"Carolyn"

Moved from index 2 to 1

Moved from index 3 to 2

- Pass a location (index) to be removed  
Moves all other elements



# Using Loops with Array Lists

- ❑ You can use the enhanced for loop with Array Lists:

```
ArrayList<String> names = . . . ;  
for (String name : names)  
{  
    System.out.println(name);  
}
```

- ❑ Or ordinary loops:

```
ArrayList<String> names = . . . ;  
for (int i = 0; i < names.size(); i++)  
{  
    String name = names.get(i);  
    System.out.println(name);  
}
```



# Working with Array Lists

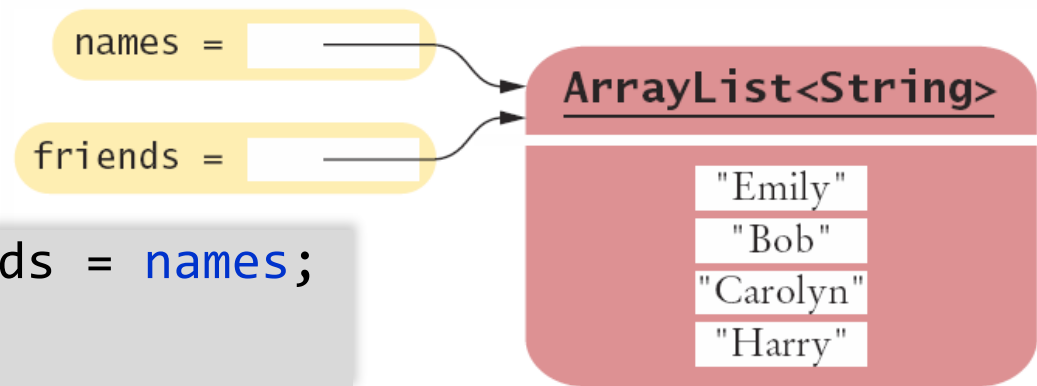
**Table 2** Working with Array Lists

<code>ArrayList&lt;String&gt; names = new ArrayList&lt;String&gt;();</code>	Constructs an empty array list that can hold strings.
<code>names.add("Ann");</code> <code>names.add("Cindy");</code>	Adds elements to the end.
<code>System.out.println(names);</code>	Prints [Ann, Cindy].
<code>names.add(1, "Bob");</code>	Inserts an element at index 1. names is now [Ann, Bob, Cindy].
<code>names.remove(0);</code>	Removes the element at index 0. names is now [Bob, Cindy].
<code>names.set(0, "Bill");</code>	Replaces an element with a different value. names is now [Bill, Cindy].
<code>String name = names.get(i);</code>	Gets an element.
<code>String last = names.get(names.size() - 1);</code>	Gets the last element.



# Copying an ArrayList

- Remember that ArrayList variables hold a reference to an ArrayList (just like arrays)
- Copying a reference:



- To copy the reference:

```
ArrayList<String> friends = names;  
friends.add("Harry");
```

reference

```
ArrayList<String> newNames = new ArrayList<String>(names);
```



# Array Lists and Methods

- ❑ Like arrays, Array Lists can be method parameter variables and return values.
- ❑ Here is an example: a method that receives a list of Strings and returns the reversed list.

reference

```
public static ArrayList<String> reverse(ArrayList<String> names)
{
    // Allocate a list to hold the method result
    ArrayList<String> result = new ArrayList<String>();
    // Traverse the names list in reverse order (last to first)
    for (int i = names.size() - 1; i >= 0; i--)
    {
        // Add each name to the result
        result.add(names.get(i));
    }
    return result;
}
```



# Wrappers and Auto-boxing

- ❑ Java provides *wrapper* classes for primitive types
  - Conversions are automatic using **auto-boxing**
    - Primitive to wrapper Class

```
double x = 29.95;  
Double wrapper;  
wrapper = x; // boxing
```

wrapper =

Double

value = 29.95

```
double x;  
Double wrapper = 29.95;  
x = wrapper; // unboxing
```

Primitive Type	Wrapper Class
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short



# Wrappers and Auto-boxing

- ❑ You cannot use primitive types in an ArrayList, but you can use their wrapper classes
  - Depend on auto-boxing for conversion
- ❑ Declare the ArrayList with wrapper classes for primitive types
  - Use ArrayList<Double>
    - Add primitive double variables
    - Or double values

```
double x = 19.95;  
ArrayList<Double> values = new ArrayList<Double>();  
values.add(29.95);           // boxing  
values.add(x);               // boxing  
double x = values.get(0);    // unboxing
```





# ArrayList Algorithms

- Converting from Array to ArrayList requires changing:

- index usage: `[i]`
- `values.length`

- To

- methods: `get()`
- `values.size()`

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

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



# Choosing Arrays or Array Lists

## ❑ Use an Array if:

- The size of the array never changes
- You have a long list of primitive values
  - For efficiency reasons
- Your instructor wants you to

## ❑ Use an Array List:

- For just about all other cases
- Especially if you have an unknown number of input values



# Array and Array List Operations

**Table 3** Comparing Array and Array List Operations

Operation	Arrays	Array Lists
Get an element.	<code>x = values[4];</code>	<code>x = values.get(4)</code>
Replace an element.	<code>values[4] = 35;</code>	<code>values.set(4, 35);</code>
Number of elements.	<code>values.length</code>	<code>values.size()</code>
Number of filled elements.	<code>currentSize</code> (companion variable, see Section 6.1.3)	<code>values.size()</code>
Remove an element.	See Section 6.3.6	<code>values.remove(4);</code>
Add an element, growing the collection.	See Section 6.3.7	<code>values.add(35);</code>
Initializing a collection.	<code>int[] values = { 1, 4, 9 };</code>	No initializer list syntax; call <code>add</code> three times.



# Common Error 6.4



## □ Length versus Size

- Unfortunately, the Java syntax for determining the number of elements in an array, an ArrayList, and a String is not consistent.
- It is a common error to confuse these. You just have to remember the correct syntax for each data type.

Data Type	Number of Elements
Array	<code>a.length</code>
Array list	<code>a.size()</code>
String	<code>a.length()</code>



# Summary: Arrays

- ❑ An array collects a sequence of values of the same type.
- ❑ Individual elements in an array values are accessed by an integer index `i`, using the notation `values[i]`.
- ❑ An array element can be used like any variable.
- ❑ An array index must be at least zero and less than the size of the array.
- ❑ A bounds error, which occurs if you supply an invalid array index, can cause your program to terminate.



# Summary: Arrays

- ❑ Use the expression `array.length` to find the number of elements in an array.
- ❑ An array reference specifies the location of an array.
- ❑ Copying the reference yields a second reference to the same array.
- ❑ With a partially-filled array, keep a companion variable for the current size.



# Summary: Arrays

- ❑ You can use the enhanced for loop to visit all elements of an array.
  - Use the enhanced for loop if you do not need the index values in the loop body.
- ❑ A linear search inspects elements in sequence until a match is found.
- ❑ Use a temporary variable when swapping elements.
- ❑ Use the `Arrays.copyOf` method to copy the elements of an array into a new array.
- ❑ Arrays can occur as method parameter variables and return values.



# Summary: Arrays

- ❑ By combining fundamental algorithms, you can solve complex programming tasks.
- ❑ You should be familiar with the implementation of fundamental algorithms so that you can adapt them.
- ❑ Discover algorithms by manipulating physical objects
- ❑ Use a two-dimensional array to store tabular data.
- ❑ Individual elements in a two-dimensional array are accessed by using two index values, `values[i][j]`





# Summary: Array Lists

- ❑ An Array List stores a sequence of values whose number can change.
  - The ArrayList class is a generic class: `ArrayList<Type>` collects elements of the specified type.
  - Use the `size` method to obtain the current size of an array list.
  - Use the `get` and `set` methods to access an array list element at a given index.
  - Use the `add` and `remove` methods to add and remove array list elements.
- ❑ To collect numbers in Array Lists, you must use wrapper classes.