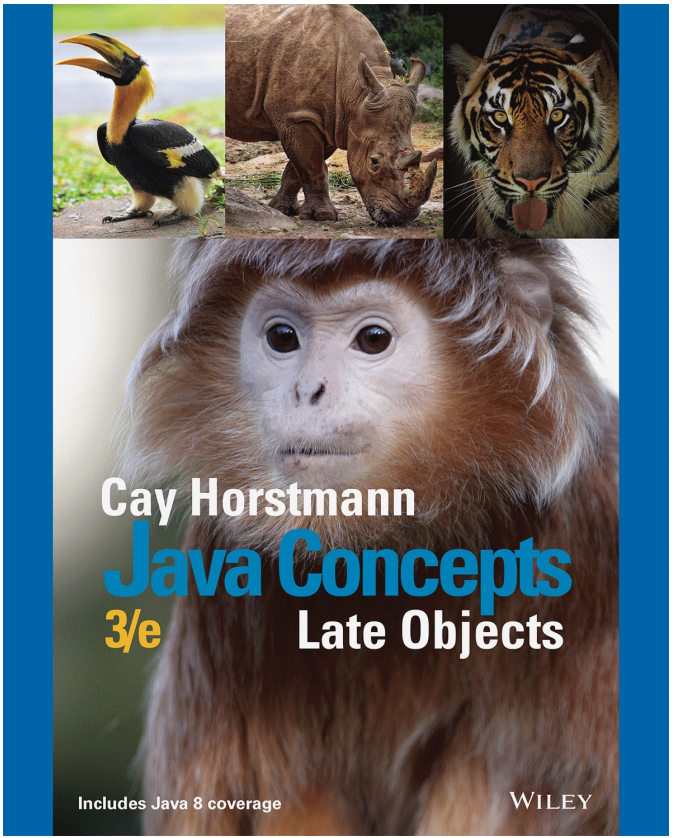
Chapter 6 – Arrays and Array Lists

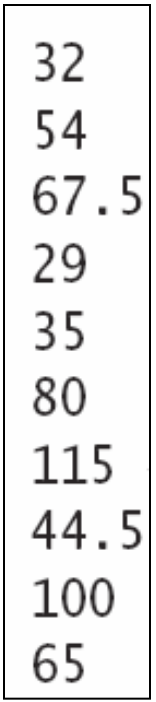


Chapter Goals

- 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

Arrays

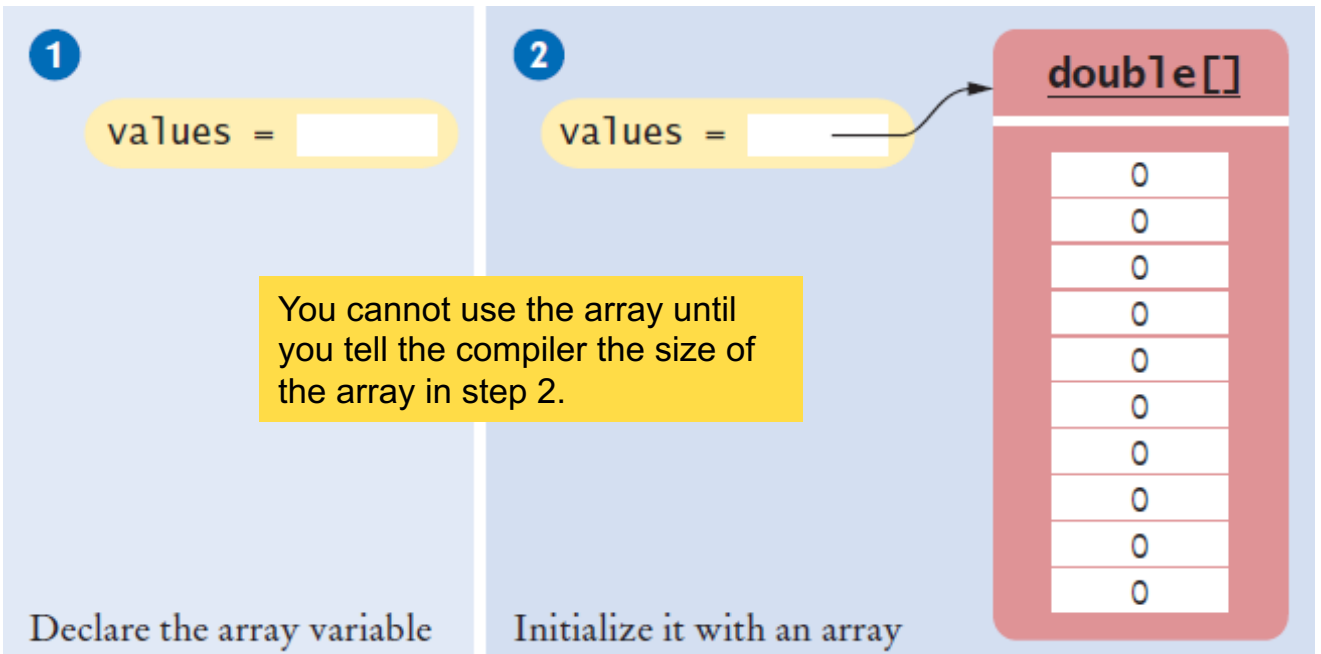
- A Computer Program often needs to store a list of values and then process them
- For example, if you had this list of values, how many variables would you need?
  - `double input1, input2, input3....`
- Arrays to the rescue!
- An array collects sequences of values of the same type



Declaring an Array

Declaring an array is a two step process

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



Declaring an Array (Step 1)

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

Type	Square Braces	Array name	Semicolon
`double`	`[ ]`	`values`	`;`

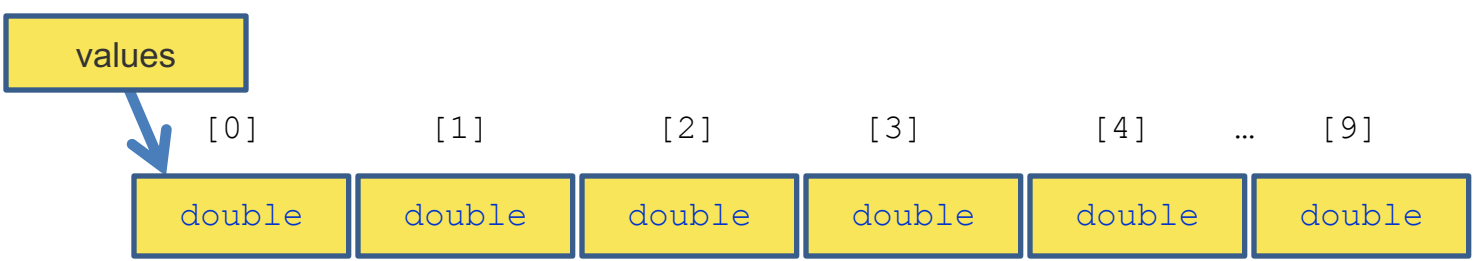
  - You are declaring that
    - There is an array named `values`
    - The elements inside are of type `double`
    - You have not (YET) declared how many elements are in inside
- Other Rules:
  - Arrays can be declared anywhere you can declare a variable
  - Do not use 'reserved' words or already used 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 without asking for a new array!



One Line Array Declaration

- Declare and Create on the same line:

Type	Braces	Array name		Keyword	Type	Size	Semicolon
`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:

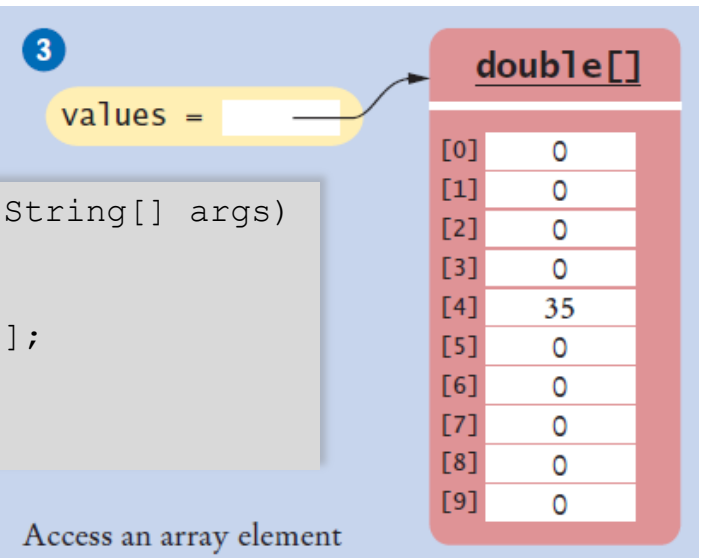
Type	Braces	Array name		Contents list	Semicolon
`int`	`[ ]`	`primes`	`=`	`{ 2, 3, 5, 7}`	`;`

- 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]`

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

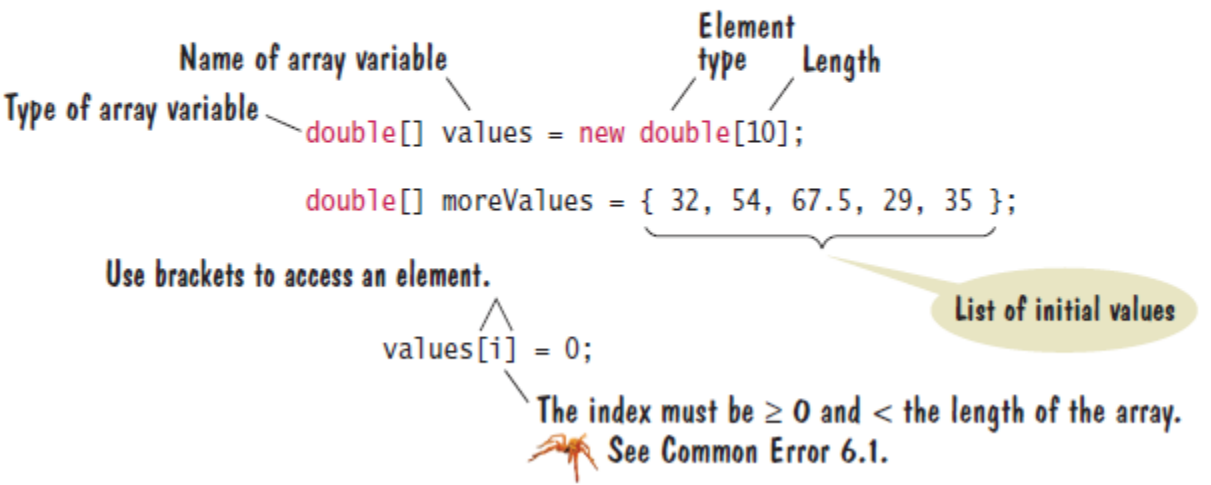


Syntax 6.1 Array

Syntax

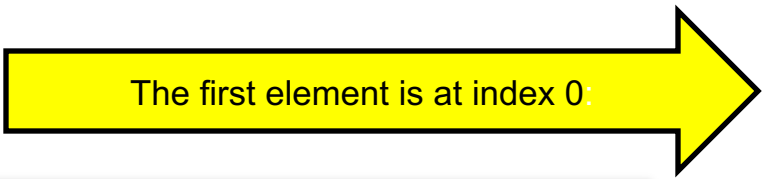
To construct an array: `new typeName[length]`

To access an element: `arrayReference[index]`

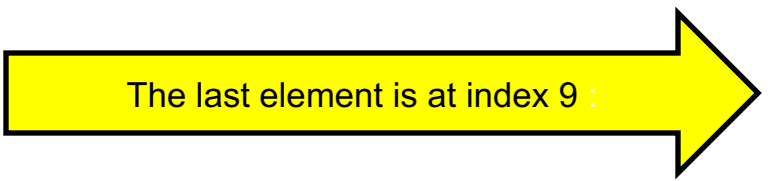


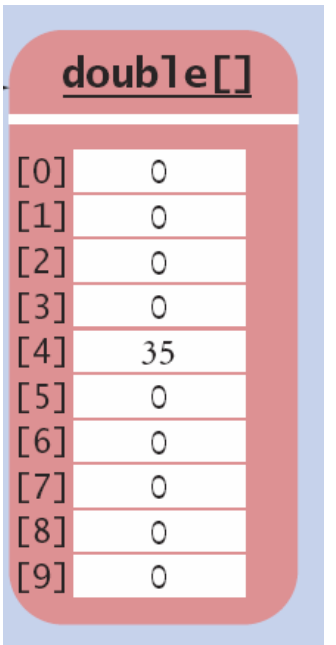
Array Index Numbers

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




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





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

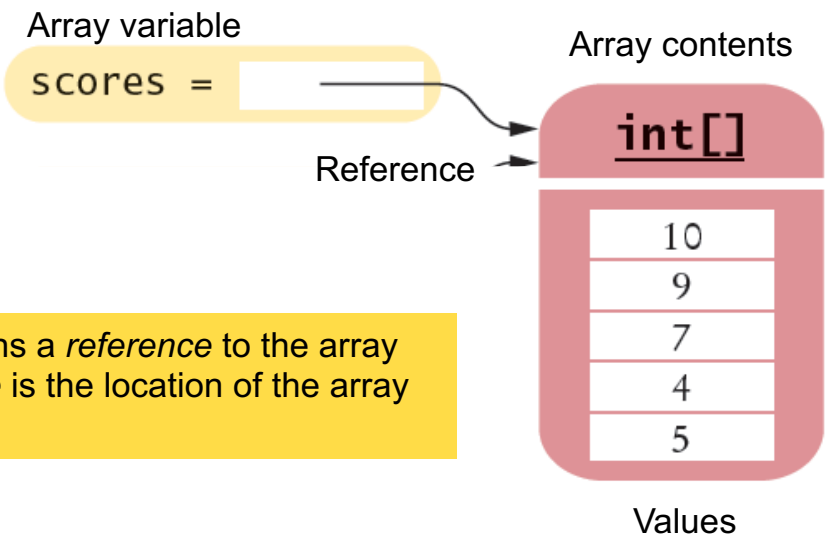
Table 1 Declaring Arrays

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

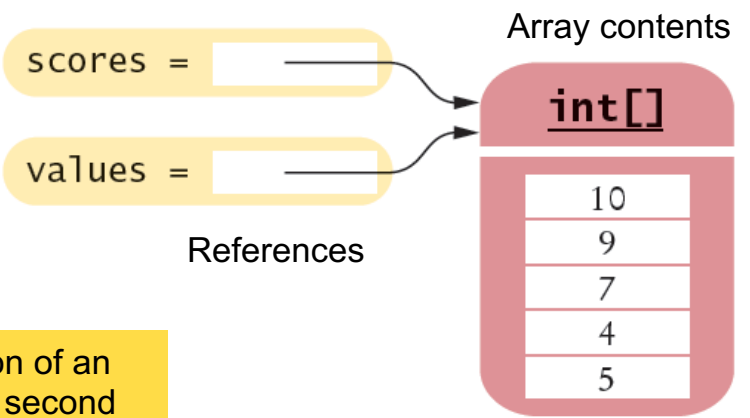


An array variable contains a *reference* to the array contents. The *reference* is the location of the array contents (in memory).

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++;
  }
}
```

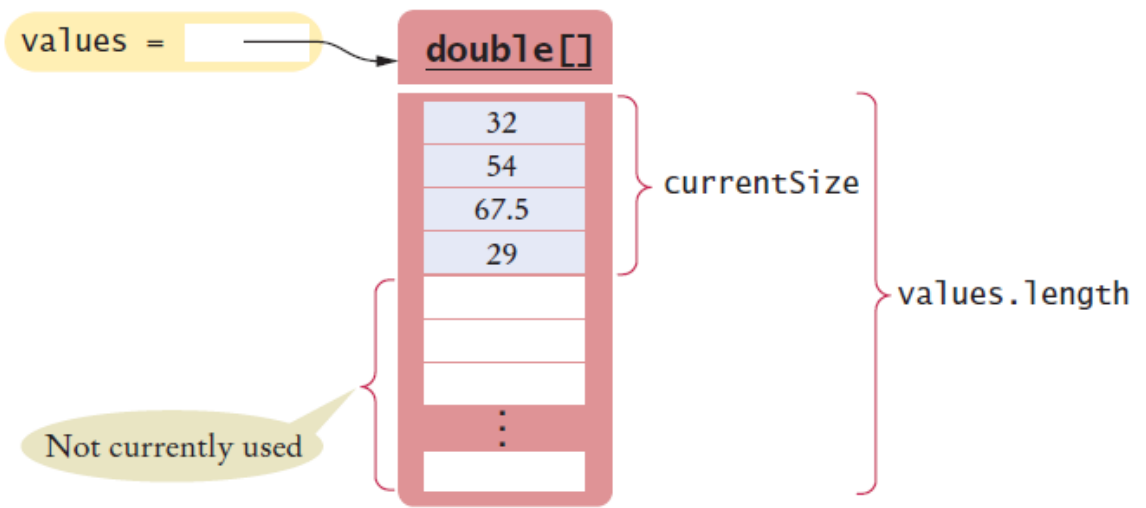
Maintain the number of elements filled using a variable (`currentSize` in this example)

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.



Self Check 6.1

Declare an array of integers containing the first five prime numbers.

Answer: `int[] primes = { 2, 3, 5, 7, 11 };`

Self Check 6.2

Assume the array `primes` has been initialized as described in Self Check 1. What does it contain after executing the following loop?

```
for (int i = 0; i < 2; i++)
{
   primes[4 - i] = primes[i];
}
```

Answer: `2, 3, 5, 3, 2`

Self Check 6.3

Assume the array `primes` has been initialized as described in Self Check 1. What does it contain after executing the following loop?

```
for (int i = 0; i < 5; i++)
{
   primes[i]++;
}
```

Answer: `3, 4, 6, 8, 12`

Self Check 6.4

Given the declaration

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

write statements to put the integer 10 into the elements of the array `values` with the lowest and the highest valid index.

Answer:

```
values[0] = 10;
values[9] = 10;
```

or better: `values[values.length - 1] = 10;`

Self Check 6.5

Declare an array called `words` that can hold ten elements of type `String`.

Answer:

```
String[] words = new String[10];
```

Self Check 6.6

Declare an array containing two strings, "`Yes`", and "`No`".

Answer: `String[] words = { "Yes", "No" };`

Self Check 6.7

Can you produce the output on page 262 without storing the inputs in an array, by using an algorithm similar to that for finding the maximum in Section 4.7.5?

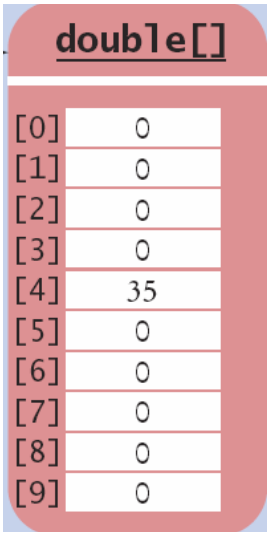
Answer: No. Because you don't store the values, you need to print them when you read them. But you don't know where to add the <= until you have seen all values.

Common Error

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

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

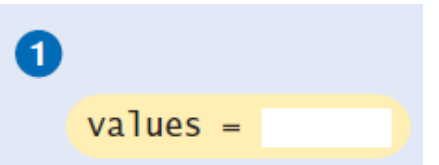
The is no element 10




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

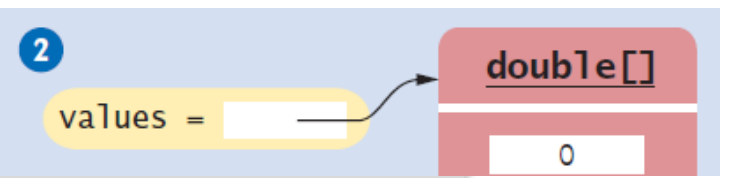
Common Error

- 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
```




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

The Enhanced for Loop

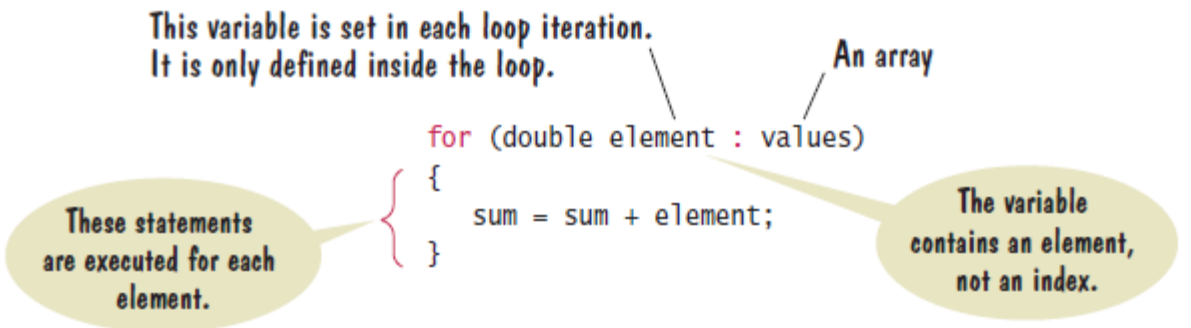
- Using `for` loops to 'walk' arrays is very common
  - The enhanced `for` loop simplifies the process
  - Also called the "for each" loop
  - Read this code as:
    - "For each element in the array"
- As the loop proceeds, it will:
  - Access each element in order (0 to length-1)
  - Copy it to the `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

Syntax

```
for (typeName variable : collection)
{
   statements
}
```



Self Check 6.8

What does this enhanced `for` loop do?

```
int counter = 0;
for (double element : values)
{
   if (element == 0) { counter++; }
}
```

Answer: It counts how many elements of `values` are zero.

Self Check 6.9

Write an enhanced `for` loop that prints all elements in the array `values`.

Answer:

```
for (double x : values)
{
   System.out.println(x);
}
```

Self Check 6.10

Write an enhanced `for` loop that multiplies all elements in a `double[]` array named `factors`, accumulating the result in a variable named `product`.

Answer:

```
double product = 1;
for (double f : factors)
{
   product = product * f;
}
```

Self Check 6.11

Why is the enhanced `for` loop not an appropriate shortcut for the following basic `for` loop?

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

Answer: The loop writes a value into `values[i]`. The enhanced `for` loop does not have the index variable `i`.

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

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

Typical `for` loop to find maximum

```
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()`
  - Useful for debugging!

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

```
[32, 54, 67.5, 29, 35]
```

Common Algorithms 5

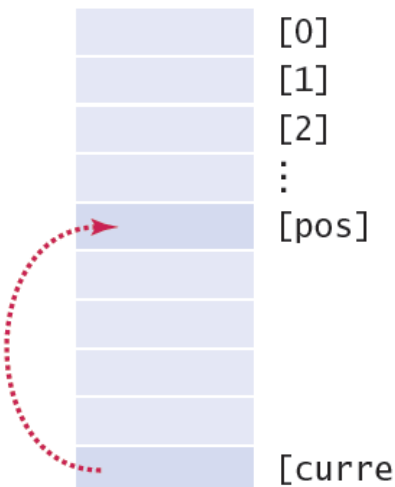
- Linear Search
  - Search for a specific value in an array
  - Start from the beginning (left), stop if/when it is found
  - Uses a boolean `found` flag to stop loop if found

```
int searchedValue = 100;  int pos = 0;
boolean found = false;
while (pos < values.length && !found)
{
  if (values[pos] == searchedValue)
  {
    found = true;
  }
  else
  {
    pos++;
}
 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

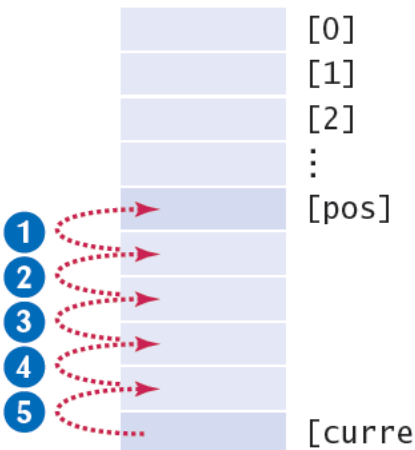
- Removing an element (at a given position)
  - Requires tracking the 'current size' (# of valid elements)
  - But don't leave a 'hole' in the array!
  - Solution depends on if you have to maintain 'order'
    - If not, find the last valid element, copy over position, update size




```
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

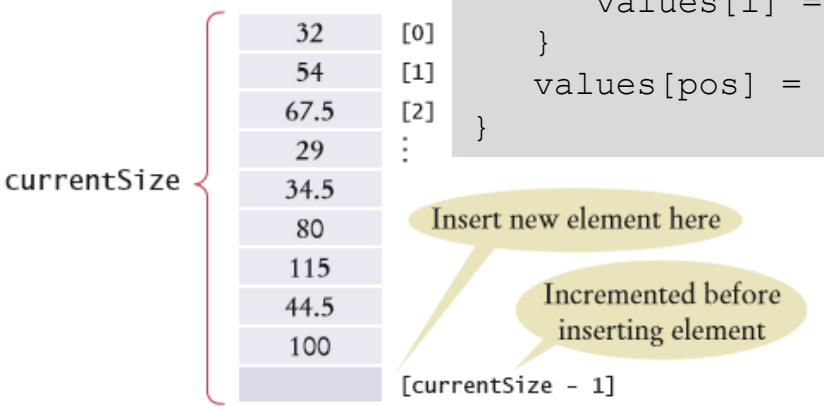



```
for (int i = pos; i < currentSize - 1; i++)
{
  values[i] = values[i + 1];
}
currentSize--;
```

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

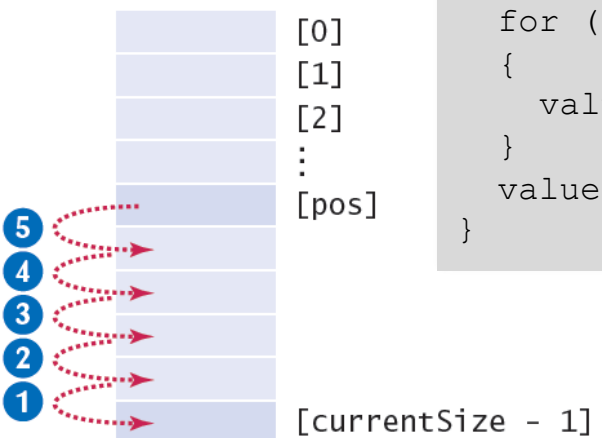
```
if (currentSize < values.length)
{  currentSize++;
   for (int i = currentSize - 1; i > pos; i--)
   {
      values[i] = values[i - 1];
   }
   values[pos] = newElement;
}
```



Common Algorithms 7

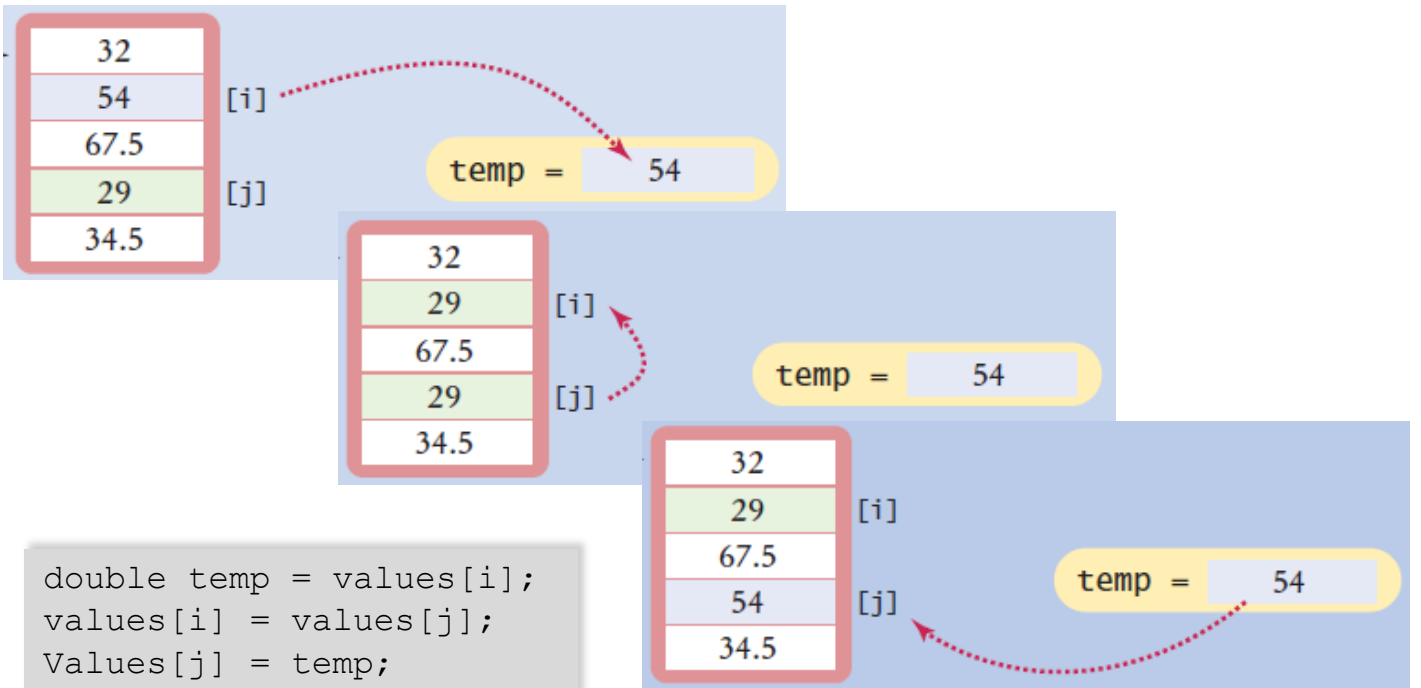
- Inserting an Element
  - Solution depends on if you have to maintain 'order'
    - If so, find the right spot for the new element, move all of the valid elements after 'pos' down one spot, insert the new element, and update size

```
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

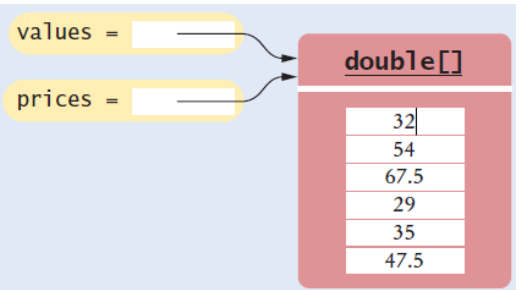
- Swapping Elements
  - Three steps using a temporary variable




```
double temp = values[i];
values[i] = values[j];
Values[j] = temp;
```

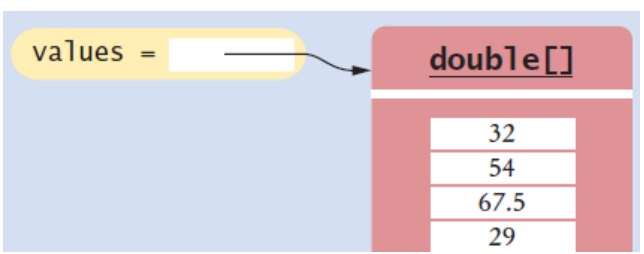
Common Algorithms 9a

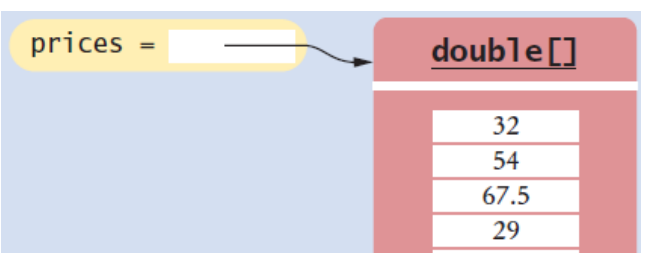
- Copying Arrays
  - Not the same as copying only the reference
    - Copying creates two set of contents!



  - Use the `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

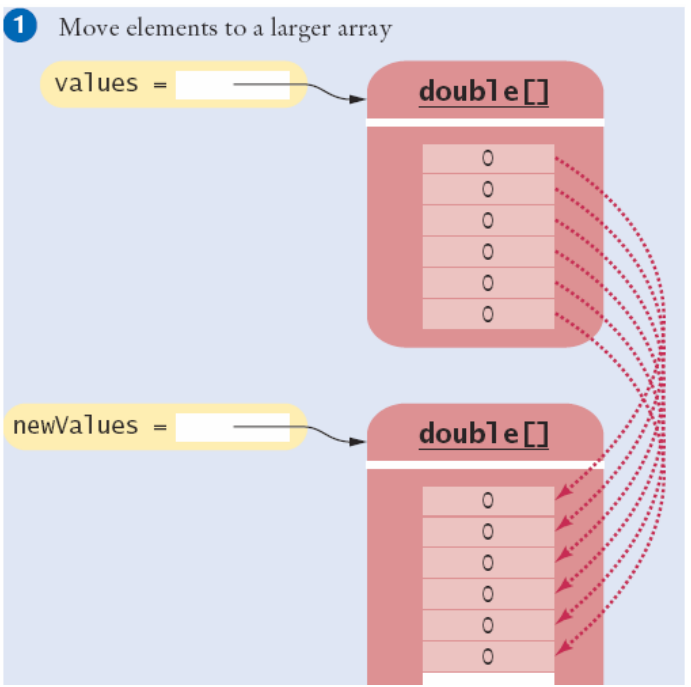
- Example: Double the size of an existing array
  - Use the `Arrays.copyOf` method
  - Use `2 *` in the second parameter

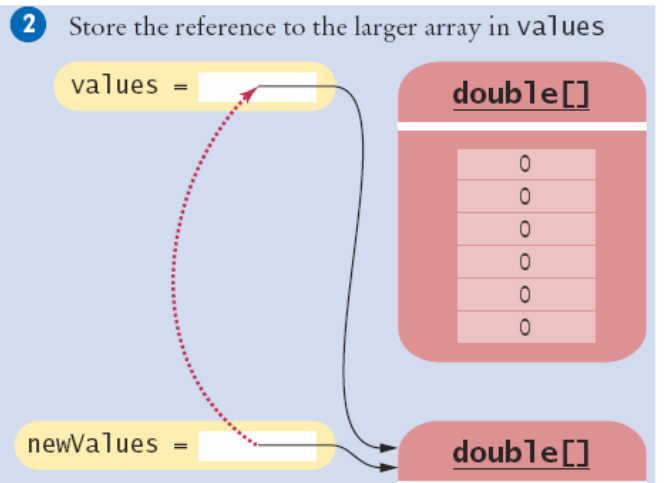
```
double[] values = new double[6];
. . . // Fill array
double[] newValues = Arrays.copyOf(values, 2 * values.length);
values = newValues;
```

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

Increasing the Size of an Array

- Copy all elements of `values` to `newValues`
- 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
```

Self Check 6.12

Given these inputs, what is the output of the `LargestInArray` program?

```
20 10 20 Q
```

Answer:

```
20 <== largest value
10
20 <== largest value
```

Self Check 6.13

Write a loop that counts how many elements in an array are equal to zero.

Answer:

```
int count = 0;
for (double x : values)
{
   if (x == 0) { count++; }
}
```

Self Check 6.14

Consider the algorithm to find the largest element in an array. Why don't we initialize `largest` and `i` with zero, like this?

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

Answer: If all elements of `values` are negative, then the result is incorrectly computed as `0`.

Self Check 6.15

When printing separators, we skipped the separator before the initial element. Rewrite the loop so that the separator is printed *after* each element, except for the last element.

Answer:

```
for (int i = 0; i < values.length; i++)
{
   System.out.print(values[i]);
   if (i < values.length - 1)
   {
      System.out.print(" | ");
   }
}
```

Now you know why we set up the loop the other way.

Self Check 6.16

What is wrong with these statements for printing an array with separators?

```
System.out.print(values[0]);
for (int i = 1; i < values.length; i++)
{
   System.out.print(", " + values[i]);
}
```

Answer: If the array has no elements, then the program terminates with an exception.

Self Check 6.17

When finding the position of a match, we used a `while` loop, not a `for` loop. What is wrong with using this loop instead?

```
for (pos = 0; pos < values.length && !found; pos++)
{
   if (values[pos] > 100)
   {
      found = true;
   }
}
```

Answer: If there is a match, then `pos` is incremented before the loop exits.

Self Check 6.18

When inserting an element into an array, we moved the elements with larger index values, starting at the end of the array. Why is it wrong to start at the insertion location, like this?

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

Answer: This loop sets all elements to `values[pos]`.

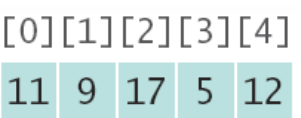
Common Error

- 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

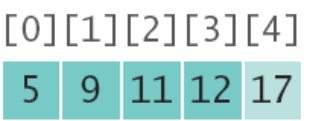
```
 Sorry, the number of lines of text is higher than
expected, and some could not be processed.  Please break
your input into smaller size segments (1000 lines maximum)
and run the program again.
```

Special Topic: Sorting Arrays

- When you store values into an array, you can choose to either:
  - Keep them unsorted (random order)



  - Sort them (Ascending or Descending…)



- 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:

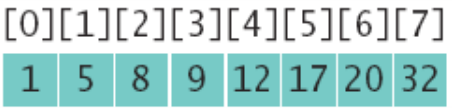
```
Arrays.sort(values);       // Sort all of the array
Arrays.sort(values, 0, currentSize);  // partially filled
```

Special Topic: Searching

- We have seen the Linear Search (6.3.5)
  - It works on an array that is sorted, or unsorted
  - Visit 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




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

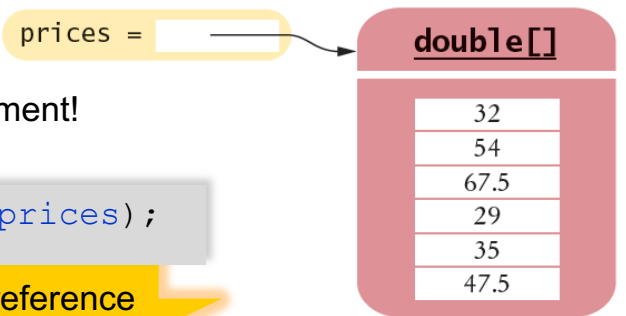
Binary Search Example

```
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); }
```

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;
}
```

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

Passing References (Step 1)

- 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

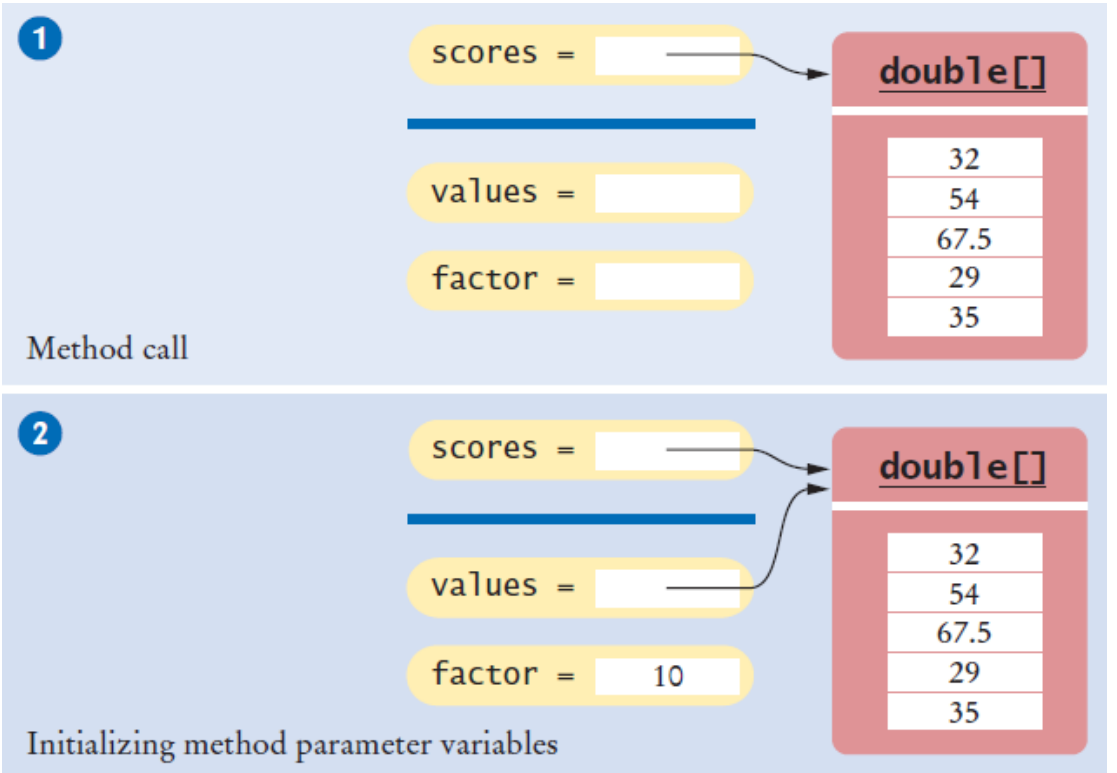
```
multiply(values,        10);
```

reference

value

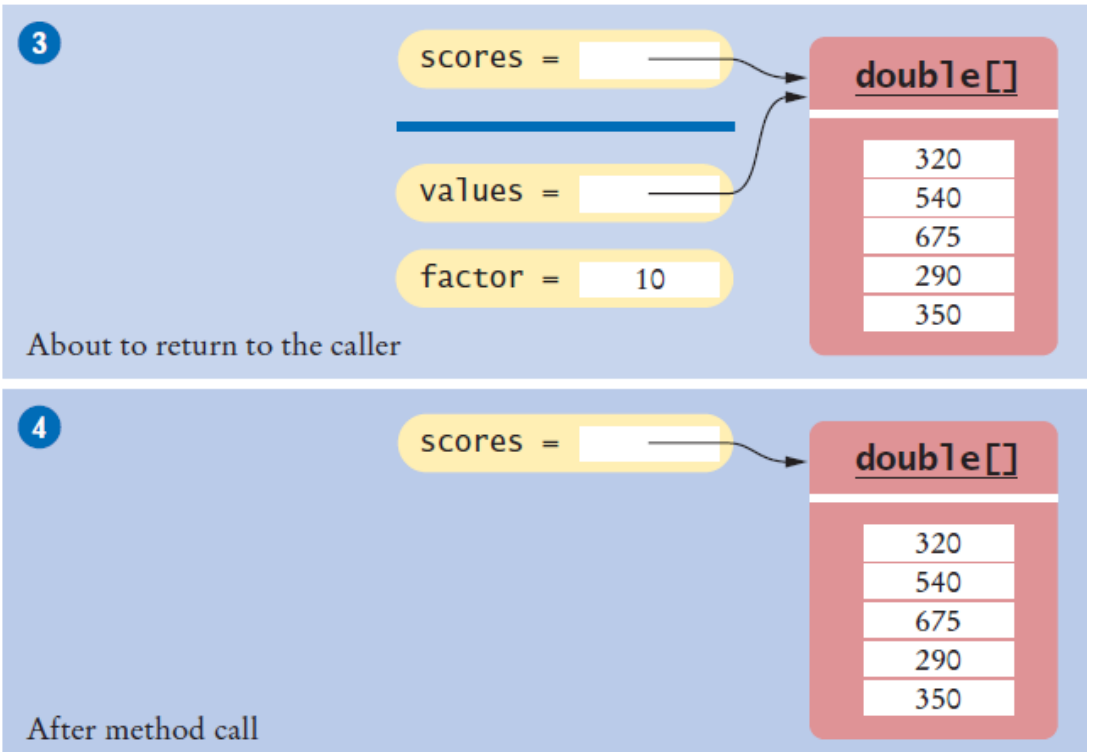
```
public static void multiply(double[] data, double factor)
{
  for (int i = 0; i < data.length; i++)
    data[i] = data[i] * factor;
}
```

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

Self Check 6.19

How do you call the `squares` method to compute the first five squares and store the result in an array `numbers`?

Answer: `int[] numbers = squares(5);`

Self Check 6.20

Write a method `fill` that fills all elements of an array of integers with a given value. For example, the call `fill(scores, 10)` should fill all elements of the array `scores` with the value 10.

Answer:

```
public static void fill(int[] values, int value)
{
   for (int i = 0; i < values.length; i++)
   {
   values[i] = value;
   }
}
```

Self Check 6.21

Describe the purpose of the following method:

```
public static int[] mystery(int length, int n)
{
   int[] result = new int[length];
   for (int i = 0; i < result.length; i++)
   {
      result[i] = (int) (n * Math.random());
   }
   return result;
}
```

Answer: The method returns an array whose length is given in the first argument. The array is filled with random integers between 0 and n - 1.

Self Check 6.22

Consider the following method that reverses an array:

```
public static int[] reverse(int[] values)
{
   int[] result = new int[values.length];
   for (int i = 0; i < values.length; i++)
   {
      result[i] = values[values.length - 1 - i];
   }
   return result;
}
```

Suppose the `reverse` method is called with an array scores that contains the numbers 1, 4, and 9. What is the contents of `scores` after the method call?

Answer: The contents of `scores` is unchanged. The `reverse` method returns a new array with the reversed numbers.

Self Check 6.23

Provide a trace diagram of the `reverse` method when called with an array that contains the values 1, 4, and 9.

Answer:

values	result	i
[1, 4, 9]	~~[0, 0, 0]~~	~~0~~
	~~[9, 0, 0]~~	~~1~~
	~~[9, 4, 0]~~	2
	[9, 4, 1]	

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

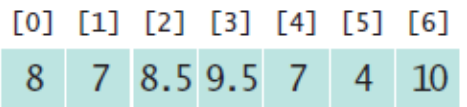
- What steps will we need?
  - Find the minimum.
  - Remove it from the array.
  - Calculate the sum.

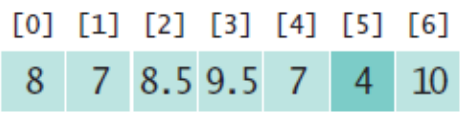
[0]	[1]	[2]	[3]	[4]	[5]	[6]
8	7	8.5	9.5	7	4	10

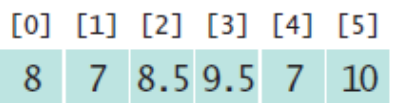
- What tools do we know?
  - Finding the minimum value (Section 6.3.3)
  - Removing an element (Section 6.3.6)
  - Calculating the sum (Section 6.3.2)
- But wait… We need to find the POSITION of the minimum value, not the value itself..
  - Hmmm. Time to adapt

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
  - Remove it from the array
  - Calculate the sum







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;
  }
}
```

Self Check 6.24

Section 6.3.6 has two algorithms for removing an element. Which of the two should be used to solve the task described in this section?

Answer: Use the first algorithm. The order of elements does not matter when computing the sum.

Self Check 6.25

It isn't actually necessary to *remove* the minimum in order to compute the total score. Describe an alternative.

Answer:

Find the minimum value.
Calculate the sum.
Subtract the minimum value from the sum.

Self Check 6.26

How can you print the number of positive and negative values in a given array, using one or more of the algorithms in Section 4.7?

Answer: Use the algorithm for counting matches (Section 4.7.2) twice, once for counting the positive values and once for counting the negative values.

Self Check 6.27

How can you print all positive values in an array, separated by commas?

Answer: You need to modify the algorithm in Section 6.3.4.

```
boolean first = true;
for (int i = 0; i < values.length; i++)
{
   if (values[i] > 0))
   {
      if (first) { first = false; }
      else { System.out.print(", "); }
   }
   System.out.print(values[i]);
}
```

Note that you can no longer use `i > 0` as the criterion for printing a separator.

Self Check 6.28

Consider the following algorithm for collecting all matches in an array:

```
int matchesSize = 0;
for (int i = 0; i < values.length; i++)
{
   if (values[i] fulfills the condition)
   {
      matches[matchesSize] = values[i];
      matchesSize++;
   }
}
```

How can this algorithm help you with Self Check 27?

Answer: Use the algorithm to collect all positive elements in an array, then use the algorithm in Section 6.3.4 to print the array of matches.

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 the position of the minimum.
   - Remove the element at the position.
   - 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);
```

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)	**Error**	That is not a legal input.

Problem Solving: Discovering Algorithms by Manipulating Physical Objects

- Consider this example problem: You are given an array whose size is an even number, and you are to switch the first and the second half.
- For example, if the array contains the eight numbers

- Rearrange it to:

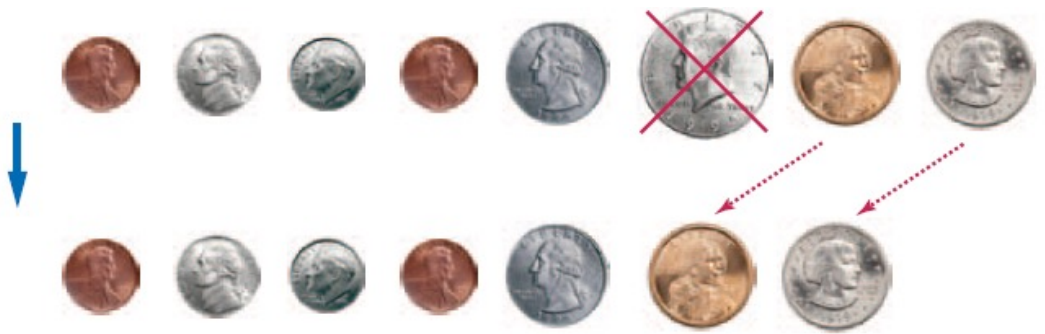
Finding the First Match

- Initialize `boolean` sentinel to `false`
- Initialize position counter to 0
  - First char in String
- Use a compound conditional in loop

Manipulating Objects

- One useful technique for discovering an algorithm is to manipulate physical objects
- Start by lining up some objects to denote an array
  - Coins, playing cards, or small toys are good choices

- Visualize removing one object



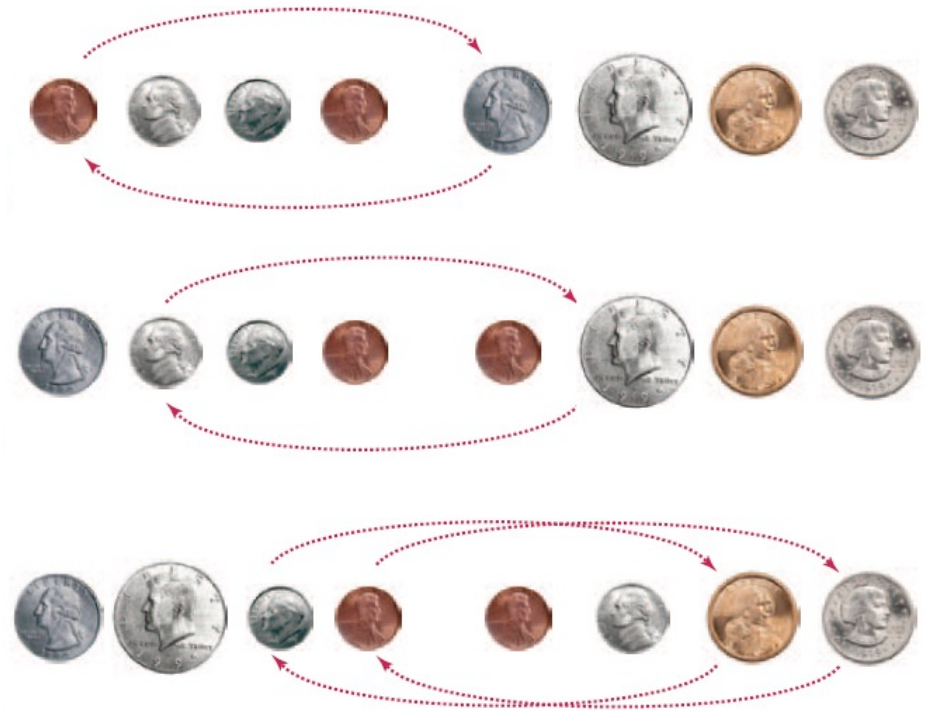
Manipulating Objects

- Visualize inserting one object

- How about swapping two coins?

Manipulating Objects

- Back to our original problem. Which tool(s) to use?
  - How about swapping two coins? Four times?



Develop an Algorithm

- Pick two locations (indexes) for the first swap and start a loop

i j

- How can j be set to handle any number of items?
  - … if size is 8, j is index 4…
- And when do we stop our loop?...

```
i = 0
j = ... (we'll think about that in a minute)
While (don't know yet)
    Swap elements at positions i and j
    i++
    j++
```

```
i = 0
j = size / 2
While (i < size / 2)
    Swap elements at positions i and j
    i++
    j++
```

Self Check 6.29

Walk through the algorithm that we developed in this section, using two paper clips to indicate the positions for i and j. Explain why there are no bounds errors in the pseudocode.

Answer: The paperclip for i assumes positions 0, 1, 2, 3. When i is incremented to 4, the condition i < size / 2 becomes false, and the loop ends. Similarly, the paperclip for j assumes positions 4, 5, 6, 7, which are the valid positions for the second half of the array.

Self Check 6.30

Take out some coins and simulate the following pseudocode, using two paper clips to indicate the positions for i and j.

```
i = 0
j = size - 1
While i < j
   Swap elements at positions i and j.
   i++
   j--
```

What does the algorithm do?

Answer: It reverses the elements in the array.

Self Check 6.31

Consider the task of rearranging all elements in an array so that the even numbers come first. Otherwise, the order doesn't matter. For example, the array

```
1 4 14 2 1 3 5 6 23
```

could be rearranged to

```
4 2 14 6 1 5 3 23 1
```

Using coins and paperclips, discover an algorithm that solves this task by swapping elements, then describe it in pseudocode.

Answer: Here is one solution. The basic idea is to move all odd elements to the end. Put one paper clip at the beginning of the array and one at the end. If the element at the first paper clip is odd, swap it with the one at the other paper clip and move that paper clip to the left. Otherwise, move the first paper clip to the right. Stop when the two paper clips meet. Here is the pseudocode:

```
i = 0
j = size - 1
While i < j
  If a[i] is odd
    Swap elements at positions i and j.
    j--
  Else
    i++
```

Self Check 6.32

Discover an algorithm for the task of Self Check 31 that uses removal and insertion of elements instead of swapping.

Answer: Here is one solution. The idea is to remove all odd elements and move them to the end. The trick is to know when to stop. Nothing is gained by moving odd elements into the area that already contains moved elements, so we want to mark that area with another paper clip.

```
i = 0
moved = size
While i < moved
  If a[i] is odd
    Remove the element at position i.
    Add the removed element to the end.
    moved--
```

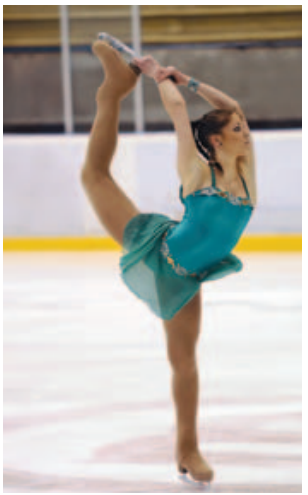
Self Check 6.33

Consider the algorithm in Section 4.7.4 that finds the largest element in a sequence of inputs—*not* the largest element in an array. Why is this algorithm better visualized by picking playing cards from a deck rather than arranging toy soldiers in a sequence?

Answer: When you read inputs, you get to see values one at a time, and you can't peek ahead. Picking cards one at a time from a deck of cards simulates this process better than looking at a sequence of items, all of which are revealed.

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

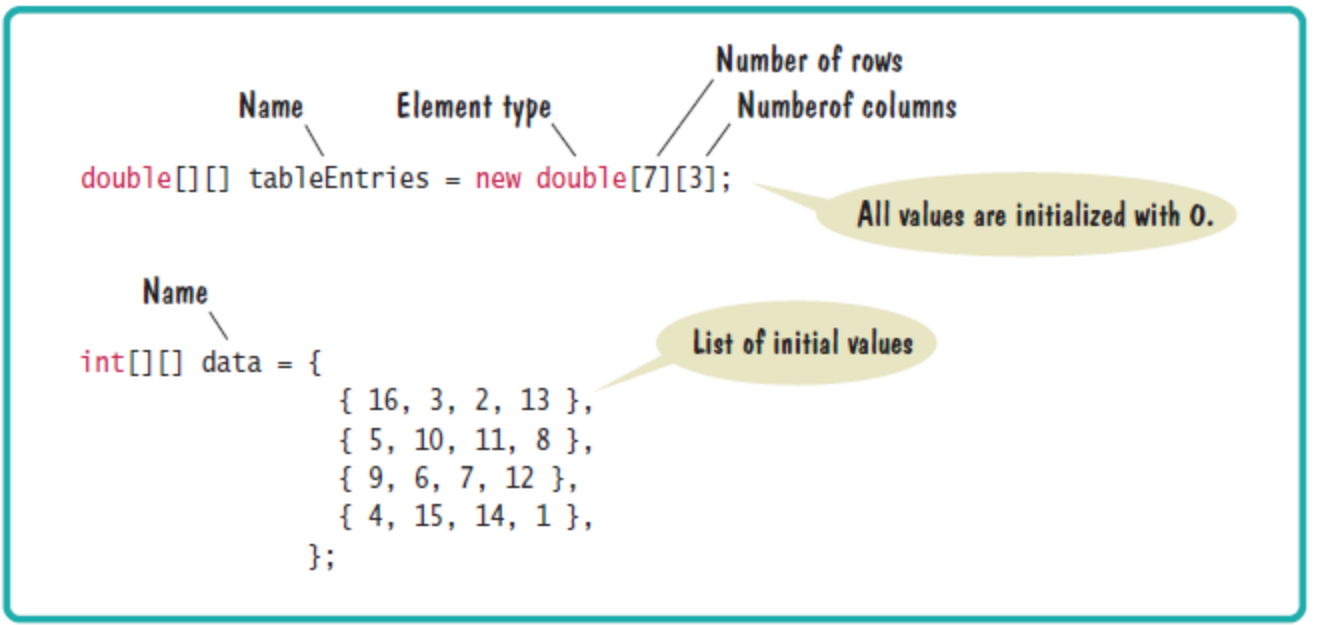
- You can also initialize the array

```
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

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



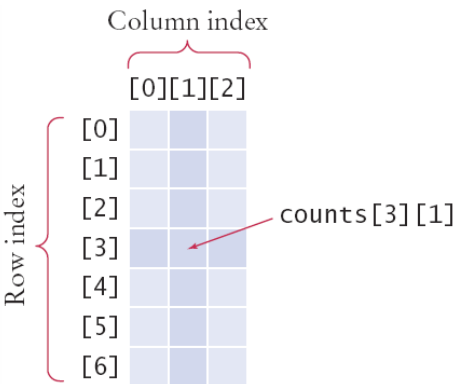
Accessing Elements

- Use two index values:

  Row then Column

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

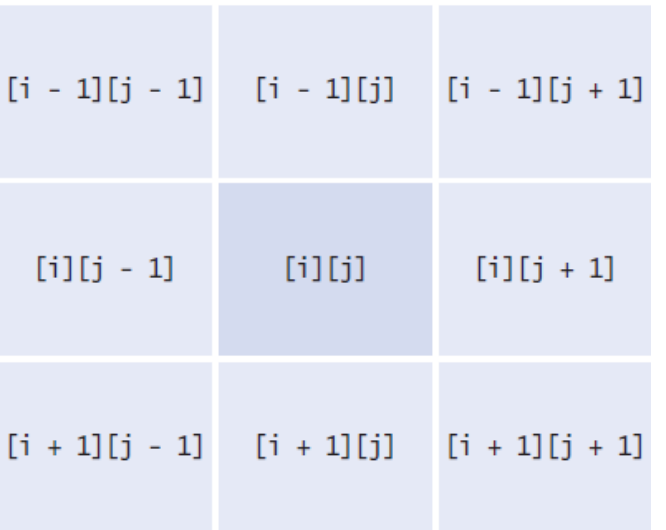
- To print
  - Use nested for loops
  - Outer row(i) , inner column(j) :




```
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

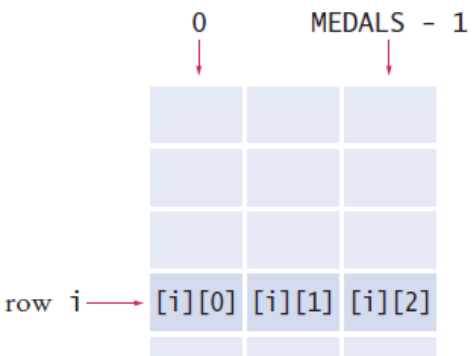
- Some programs that work with two-dimensional arrays need to locate the elements that are adjacent to an element
- This task is particularly common in games
- You are at loc `i`, `j`
- Watch out for edges!
  - No negative indexes!
  - Not off the ‘board’



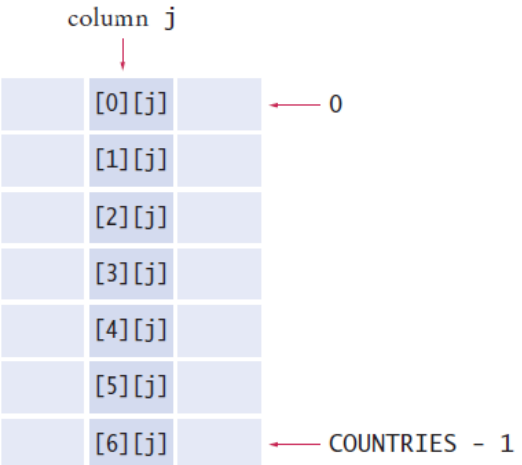
Adding Rows and Columns

Rows (x)

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



Columns (y)




```
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          };
```

```
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 new line
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
```

Self Check 6.34

What results do you get if you total the columns in our sample data?

Answer: You get the total number of gold, silver, and bronze medals in the competition. In our example, there are five of each.

Self Check 6.35

Consider an 8 × 8 array for a board game:

```
int[][] board = new int[8][8];
```

Using two nested loops, initialize the board so that zeroes and ones alternate, as on a checkerboard:

```
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
. . .
1 0 1 0 1 0 1 0
```

Hint: Check whether `i + j` is even.

Answer:

```
for (int i = 0; i < 8; i++)
{
   for (int j = 0; j < 8; j++)
   {
      board[i][j] = (i + j) % 2;
   }
}
```

Self Check 6.36

Declare a two-dimensional array for representing a tic-tac-toe board. The board has three rows and columns and contains strings "x", "o", and " ".

Answer: `String[][] board = new String[3][3];`

Self Check 6.37

Write an assignment statement to place an "x" in the upper-right corner of the tic-tac-toe board in Self Check 36.

Answer: `board[0][2] = "x";`

Self Check 6.38

Which elements are on the diagonal joining the upper-left and the lower-right corners of the tic-tac-toe board in Self Check 36?

Answer: `board[0][0], board[1][1], board[2][2]`

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.

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>();
```

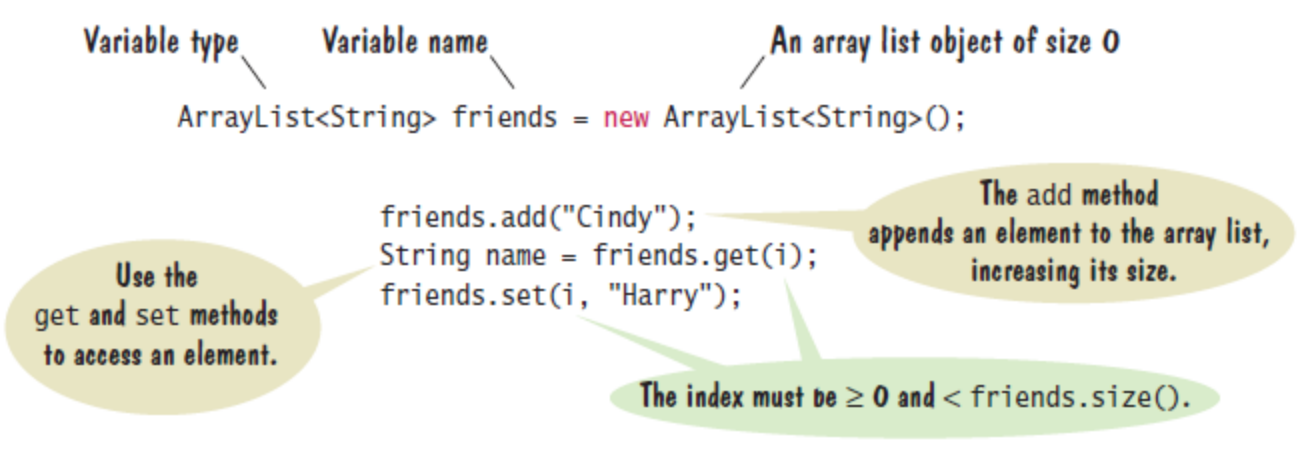
Syntax 6.4 Array Lists

- ArrayList provides many useful methods:
  - add: add an element
  - get: return an element
  - remove: delete an element
  - set: change an element
  - size: current length

Syntax

To construct an array list: `new ArrayList<typeName>()`

To access an element: `arraylistReference.get(index)`
`arraylistReference.set(index, value)`



Adding an Element with add()

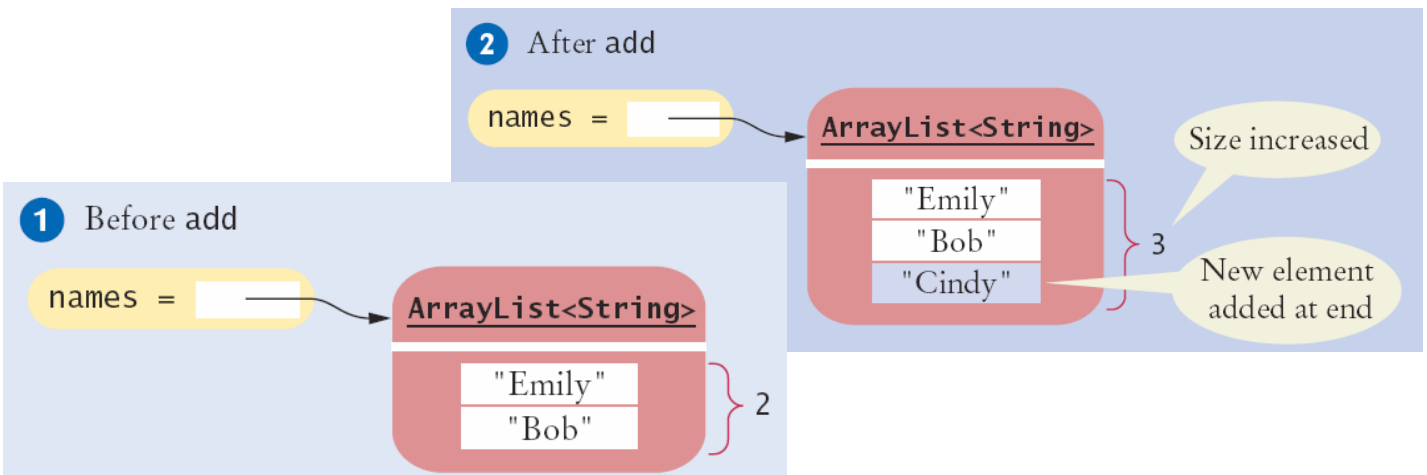
- The add method has two versions:
  - Pass a new element to add to the end

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

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

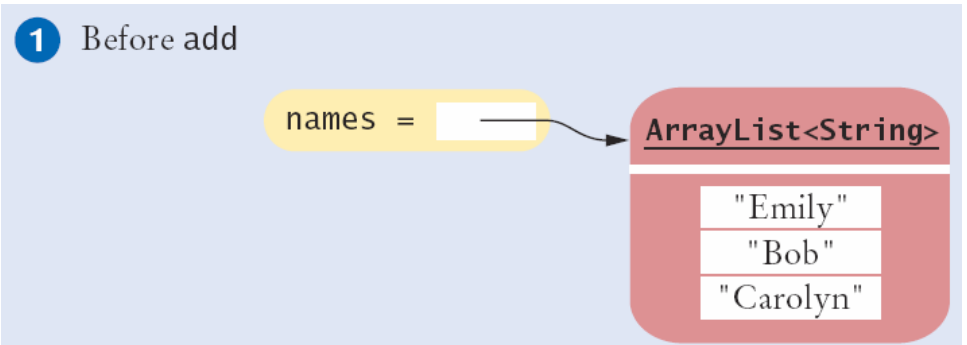
```
names.add(1,"Cindy");
```

Moves all other elements

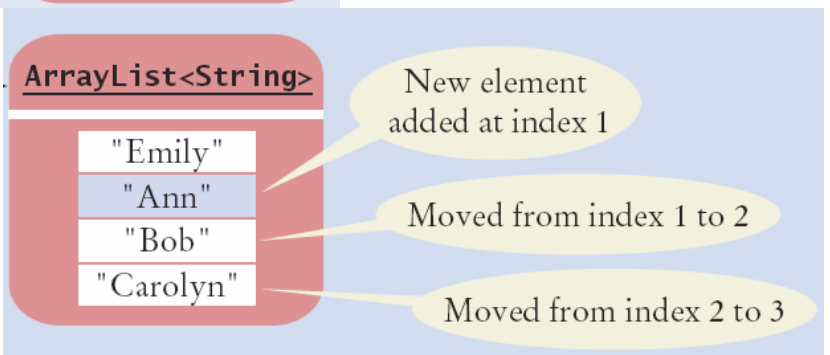


Adding an Element in the Middle

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

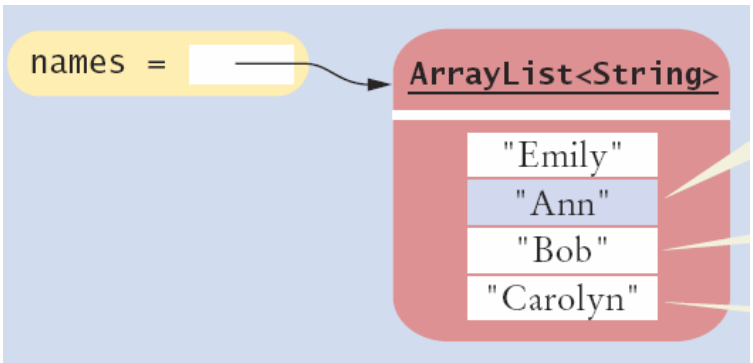



```
names.add(1, "Ann");
```

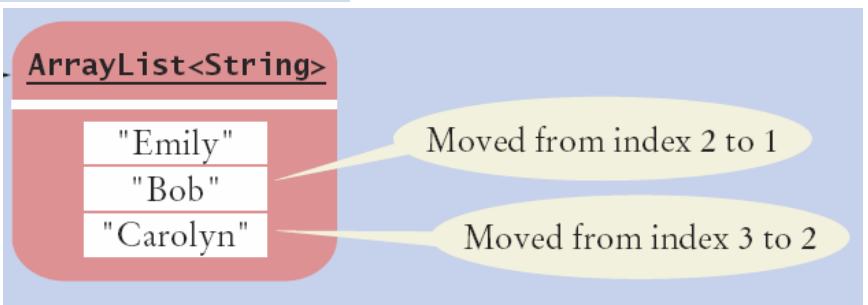


Removing an Element

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




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



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);
}
```

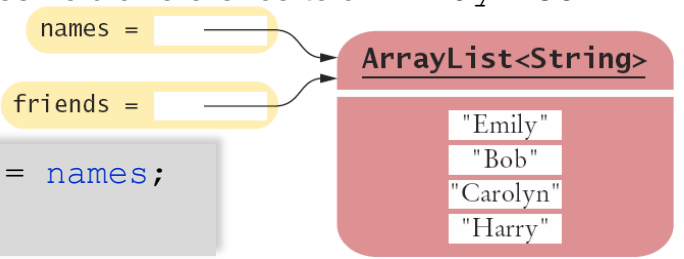
Working with Array Lists

Table 2 Working with Array Lists

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

Copying an ArrayList

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




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

- To make a copy, pass the reference of the original `ArrayList` to the constructor of the new one:

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 `String`s and returns the reversed list

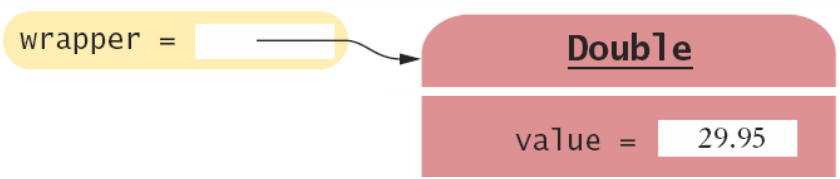
reference

```
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` Class to primitive

```
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`

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

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

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

Self Check 6.39

Declare an array list `primes` of integers that contains the first five prime numbers (2, 3, 5, 7, and 11).

Answer:

```
ArrayList<Integer> primes =
  new ArrayList<Integer>();
primes.add(2);
primes.add(3);
primes.add(5);
primes.add(7);
primes.add(11);
```

Self Check 6.40

Given the array list `primes` declared in Self Check 39, write a loop to print its elements in reverse order, starting with the last element.

Answer:

```
for (int i = primes.size() - 1; i >= 0; i--)
{
   System.out.println(primes.get(i));
}
```

Self Check 6.41

What does the array list `names` contain after the following statements?

```
ArrayList<String> names = new ArrayList<String>;
names.add("Bob");
names.add(0, "Ann");
names.remove(1);
names.add("Cal");
```

Answer: "Ann", "Cal"

Self Check 6.42

What is wrong with this code snippet?

```
ArrayList<String> names;
names.add(Bob);
```

Answer: The `names` variable has not been initialized.

Self Check 6.43

Consider this method that appends the elements of one array list to another.

```
public static void append(ArrayList<String> target,
  ArrayList<String> source)
{
   for (int i = 0; i < source.size(); i++)
   {
      target.add(source.get(i));
   }
}
```

What are the contents of names1 and names2 after these statements?

```
ArrayList<String> names1 = new ArrayList<String>();
names1.add("Emily");
names1.add("Bob");
names1.add("Cindy");
ArrayList<String> names2 = new ArrayList<String>();
names2.add("Dave");
append(names1, names2);
```

Answer: `names1` contains “Emily”, “Bob”, “Cindy”, “Dave”; names2 contains “Dave”

Self Check 6.44

Suppose you want to store the names of the weekdays. Should you use an array list or an array of seven strings?

Answer: Because the number of weekdays doesn't change, there is no disadvantage to using an array, and it is easier to initialize:

```
String[] weekdayNames = { "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };
```

Self Check 6.45

The `sec08` directory of your source code contains an alternate implementation of the problem solution in How To 6.1 on page 287. Compare the array and array list implementations. What is the primary advantage of the latter?

Answer: Reading inputs into an array list is much easier.

Common Error

- 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	`a.length`
Array list	`a.size()`
`String`	`a.length()`