LECTURE 4

ARRAYS and ARRAYLISTS

Lecture Contents

- User input
- Math.random
- Javadocs
- Arrays
- The Enhanced `for` Loop
- Using Arrays with Methods
- Two-Dimensional Arrays
- Array Lists

2.3 Input and Output

Reading Input

- You might need to ask for input (aka prompt for input) and then save what was entered.
  - We will be reading input from the keyboard
- This is a three step process in Java

1) Import the `Scanner` class from its 'package' `java.util`

```
import java.util.Scanner;
```

2) Setup an object of the `Scanner` class

```
Scanner in = new Scanner(System.in);
```

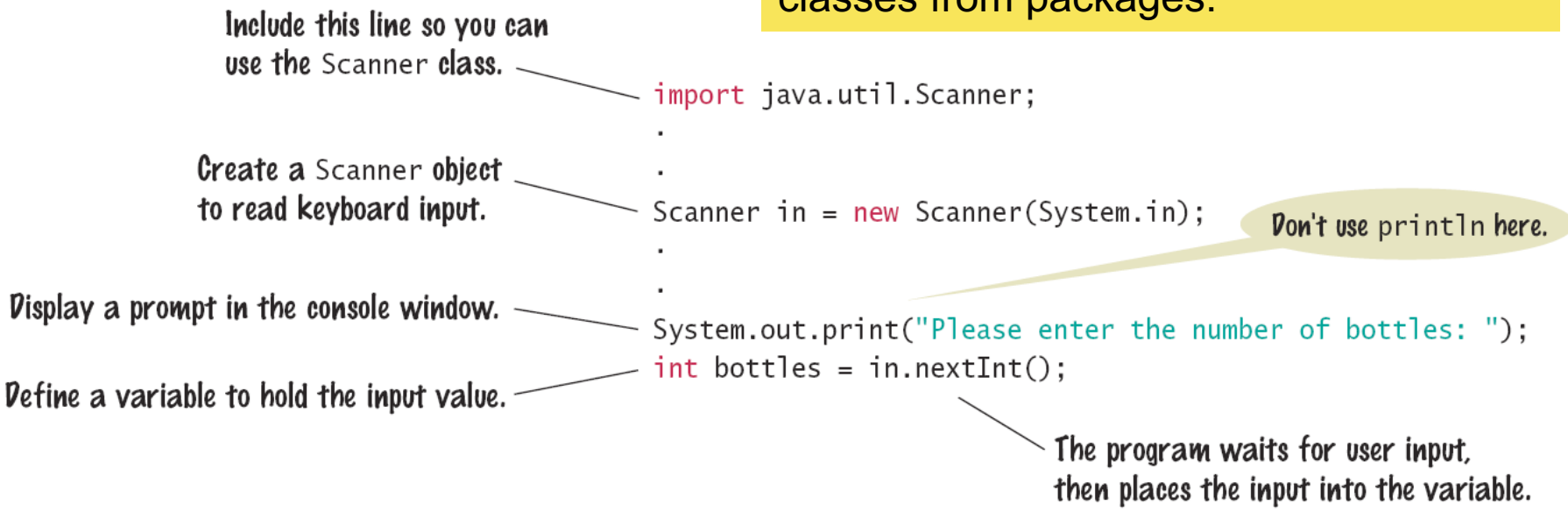
3) Use methods of the new `Scanner` object to get input

```
String name = in.next();
int bottles = in.nextInt();
double price = in.nextDouble();
```

Syntax 2.3: Input Statement

- The `Scanner` class allows you to read keyboard input from the user
  - It is part of the Java API `util` package

Java classes are grouped into packages. Use the `import` statement to use classes from packages.



RandomDemo.java

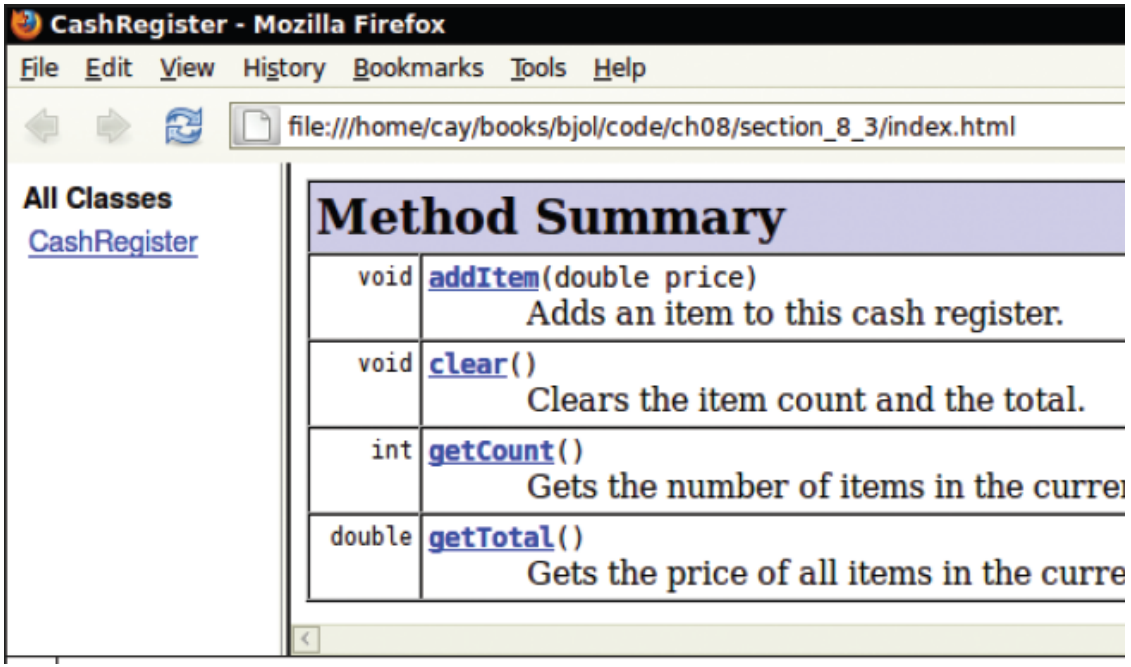
```
1  /**
2     This program prints ten random numbers between 0 and 1.
3  */
4  public class RandomDemo
5  {
6     public static void main(String[] args)
7     {
8        for (int i = 1; i <= 10; i++)
9        {
10          double r = Math.random();
11          System.out.println(r);
12       }
13    }
14 }
```

Program Run

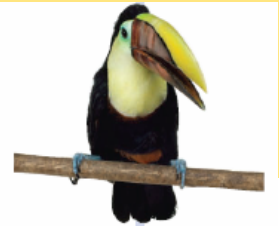
```
0.2992436267816825
0.4386017604531353 7
0.7365753471168408
0.6880250194282326
0.1608272403783395
0.5362876579988844
0.3098705906424375
0.6602909916554179
0.1927951611482942
0.8632330736331089
```

Special Topic 8.1: Javadoc

- The Javadoc utility generates a set of HTML files from the Javadoc style comments in your source code
  - Methods document parameters and returns:
    - `@param`
    - `@return`



Method Comments



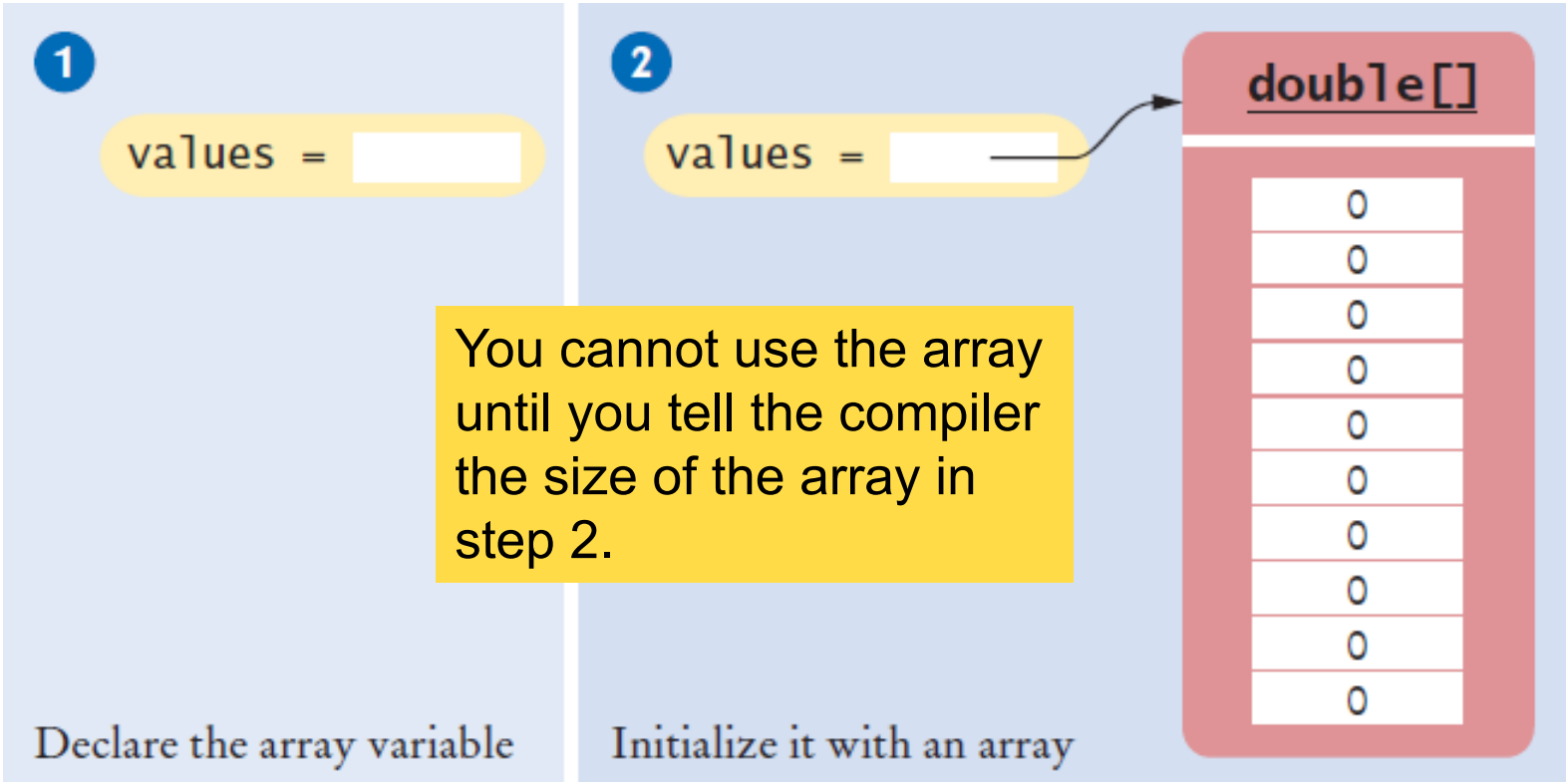
- Write a Javadoc comment above each method
- Start with `/**`
  - Note the purpose of the method
  - `@param` Describe each parameter variable
  - `@return` Describe the return value
- End with `*/`

```
/**
   Computes the volume of a cube.
   @param sideLength the side length of the cube
   @return the volume
*/
public static double cubeVolume(double sideLength)
```

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)

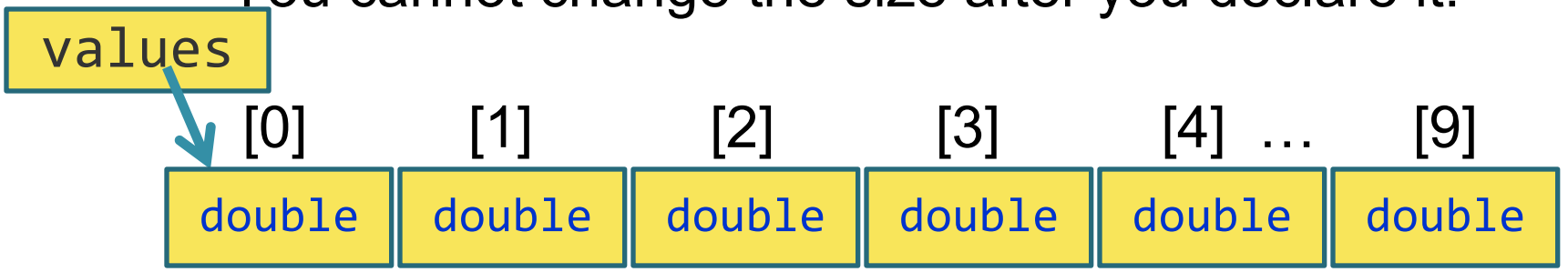
- Reserve memory for all of the elements:

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!



One Line Array Declaration

- Declare and Create on the same line:

Type	Braces	Array name		Keyword	Type	Size	semi
`double`	`[]`	`values`	`=`	`new`	`double`	`[10]`	`;`

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

    elements the size of type `double`
  - You are also setting up the array variable

Declaring and Initializing an Array

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

Type	Braces	Array name		contents list	semi
`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

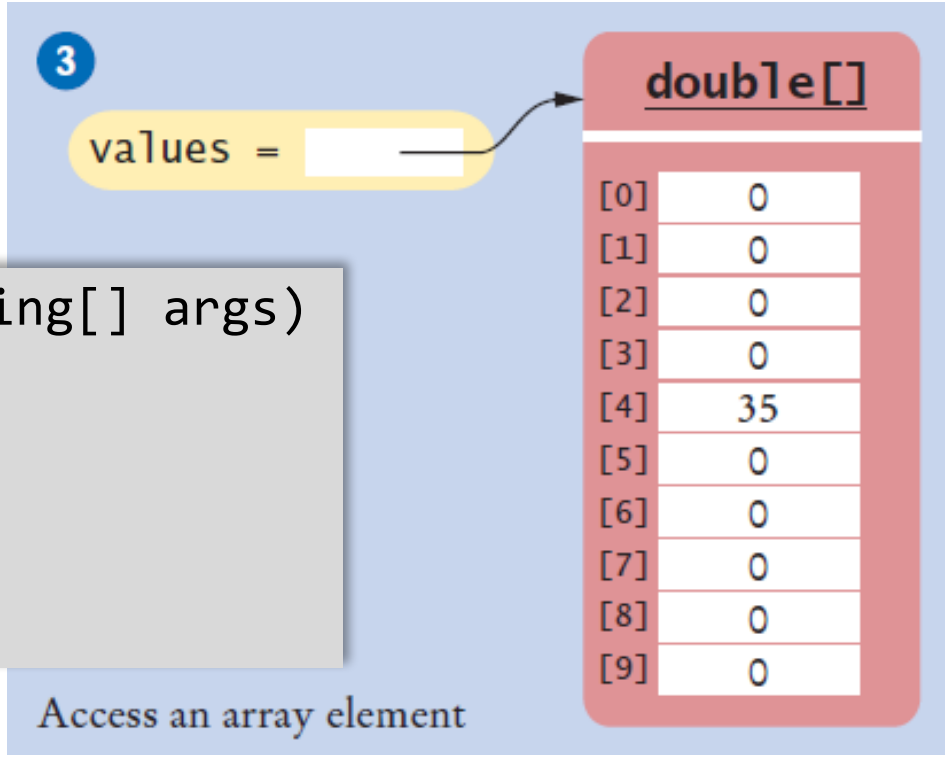
Accessing Array Elements

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

```
values[i]
```

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

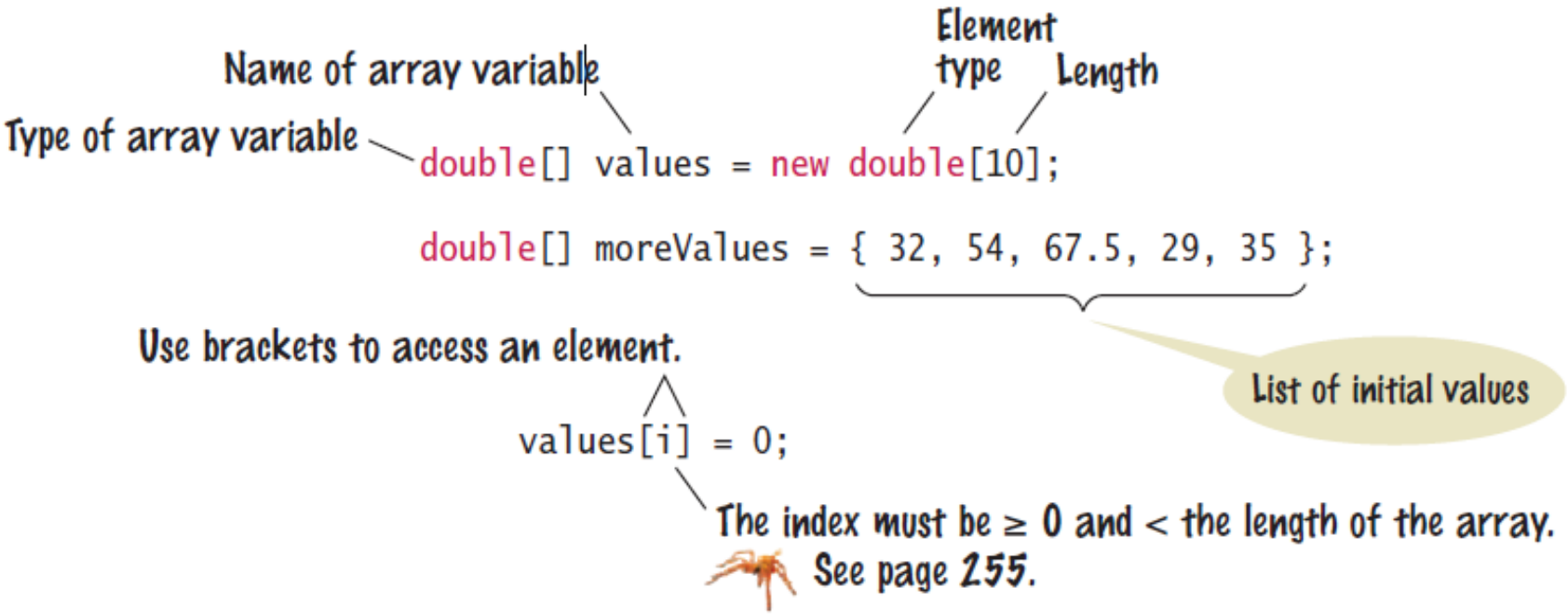
```
public static void main(String[] args)
{
   double values[];
   values = new double[10];
   values[4] = 35;
}
```



Access an array element

Syntax 6.1: Array

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



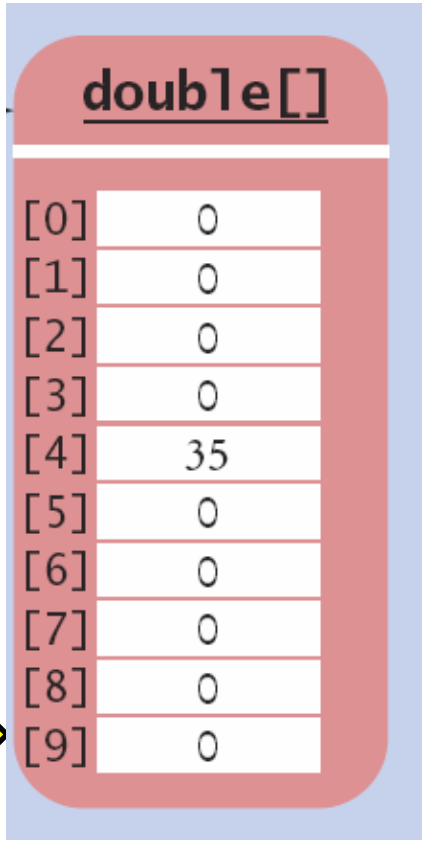
Array Index Numbers

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

The first element is at index 0:

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

The last element is at index 9:



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];
```

```
final int LENGTH = 10;
int[] numbers = new int[LENGTH];
```

```
int length = in.nextInt();
double[] data = new double[length];
```

```
int[] squares = { 0, 1, 4, 9, 16 };
```

```
String[] friends = { "Emily", "Bob", "Cindy" };
```

```
double[] data = new int[10]
```

Summary: Declaring Arrays

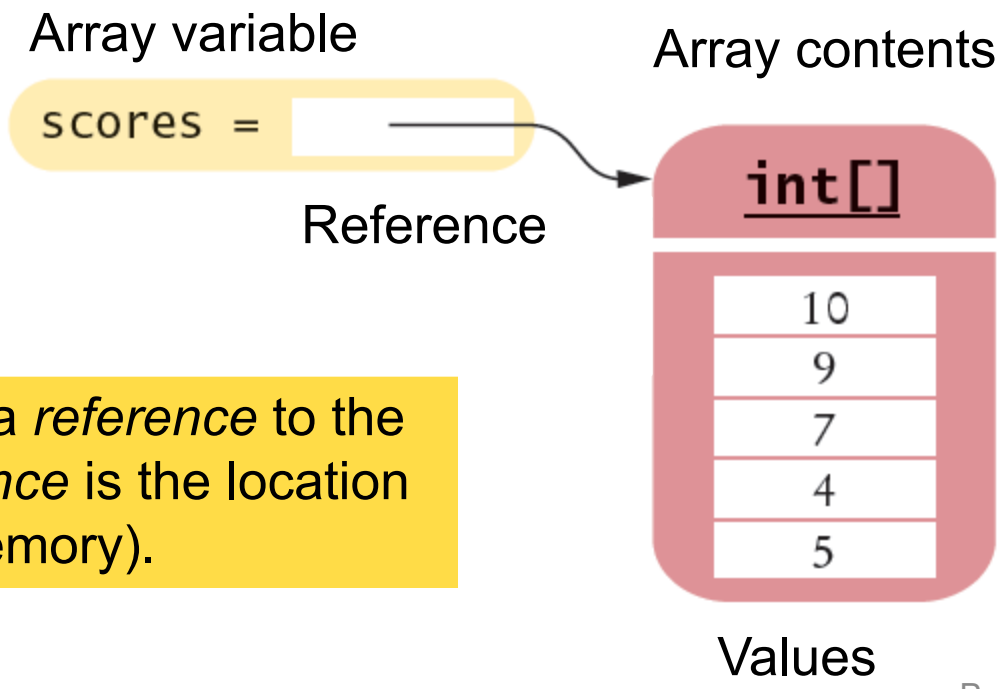
Table 1 Declaring Arrays

`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 };
```

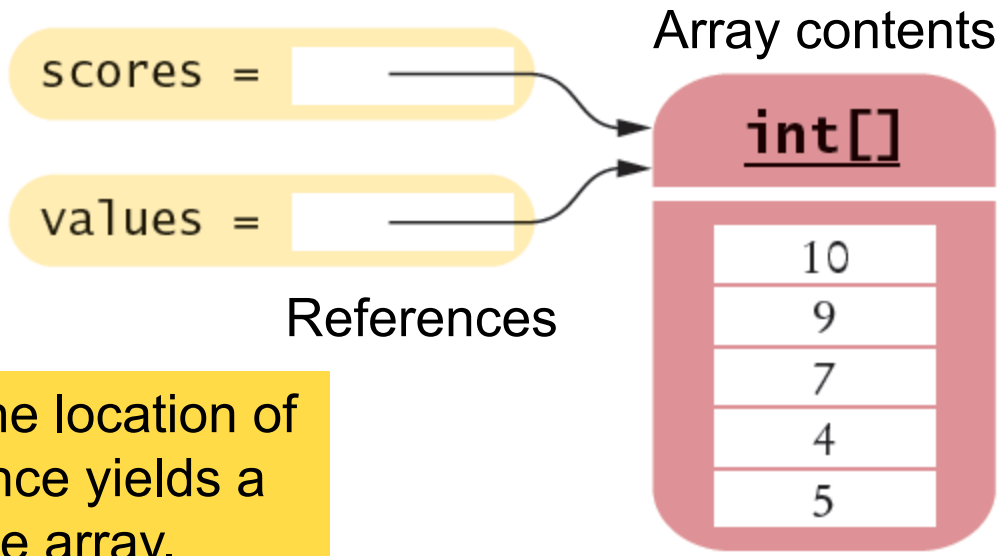


An array variable contains a *reference* to the array contents. The *reference* is the location of the array contents (in memory).

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++;
  }
}
```

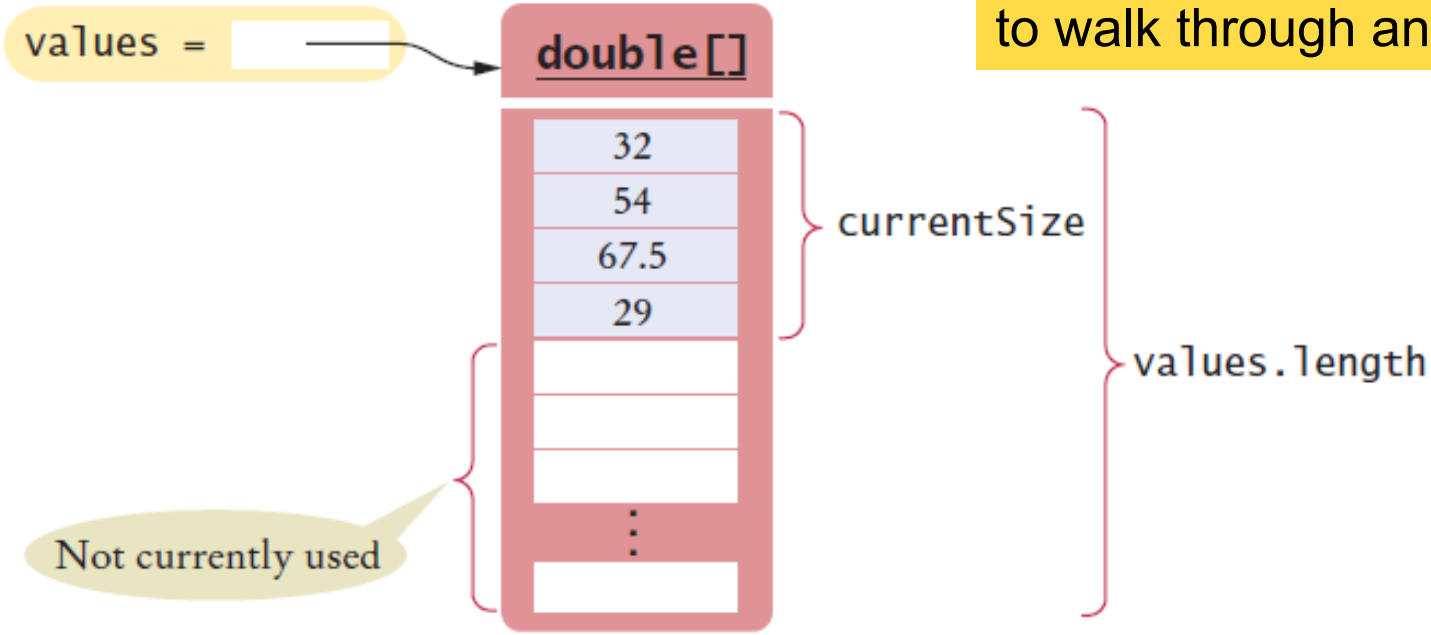
Maintain the number of elements filled using a variable (`currentSize` in this example)

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

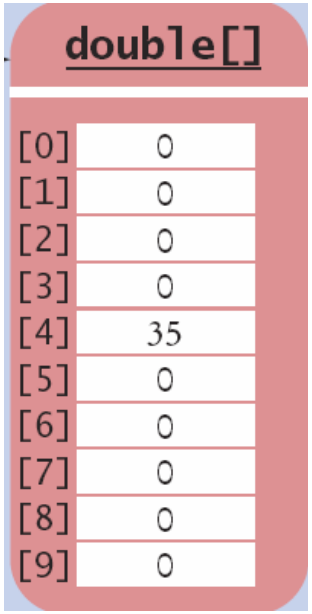


Common Error 6.1

- 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;
   }
}
```

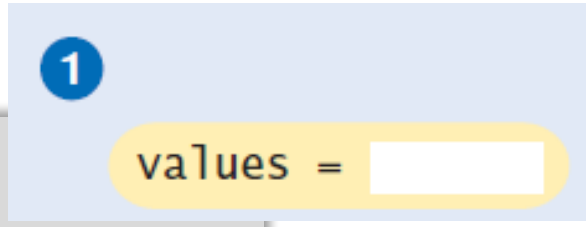
The is no element 10:




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

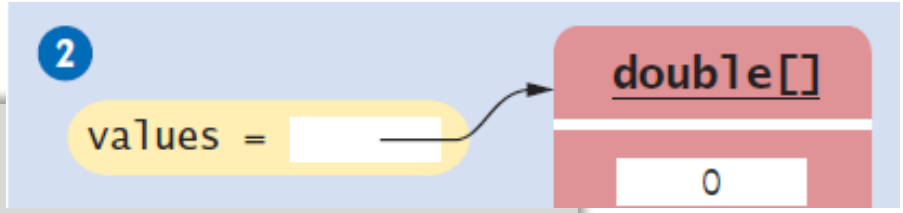
Common Error 6.2

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




```
double[] values;
...
values[0] = 29.95; // Error—values not initialized
```

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




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

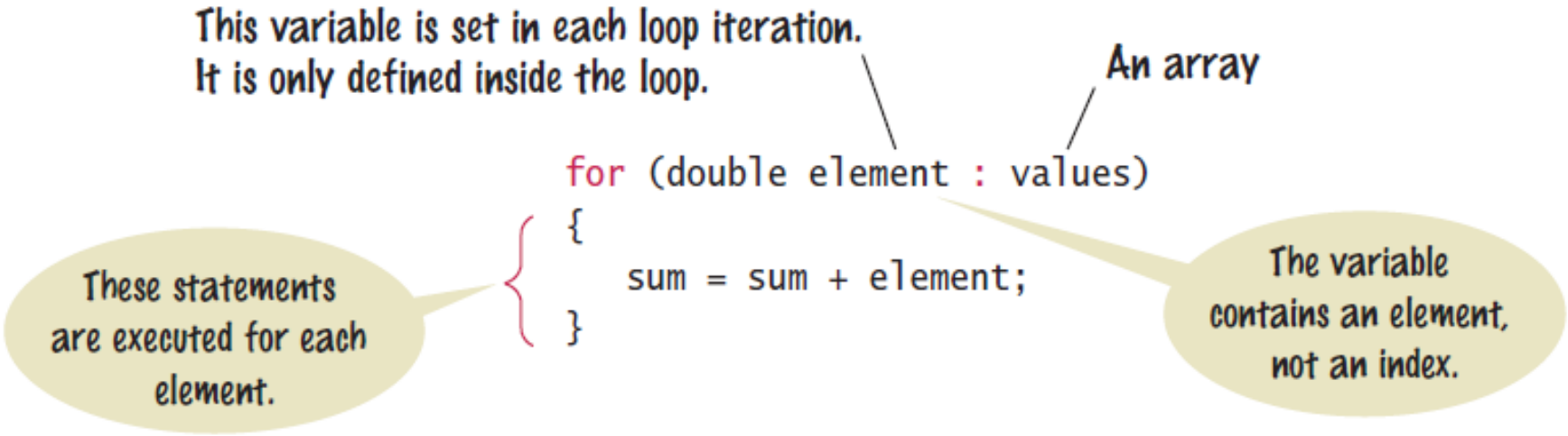
6.2 The Enhanced for Loop

- 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;
}
```

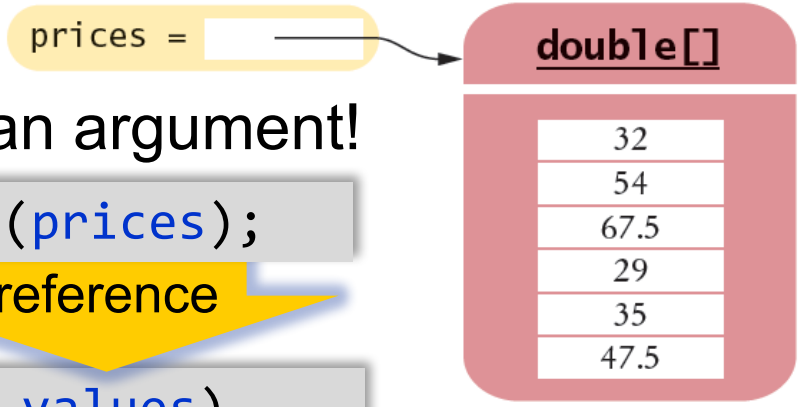
Syntax 6.2: The Enhanced `for` loop

- 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



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;
}
```

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!

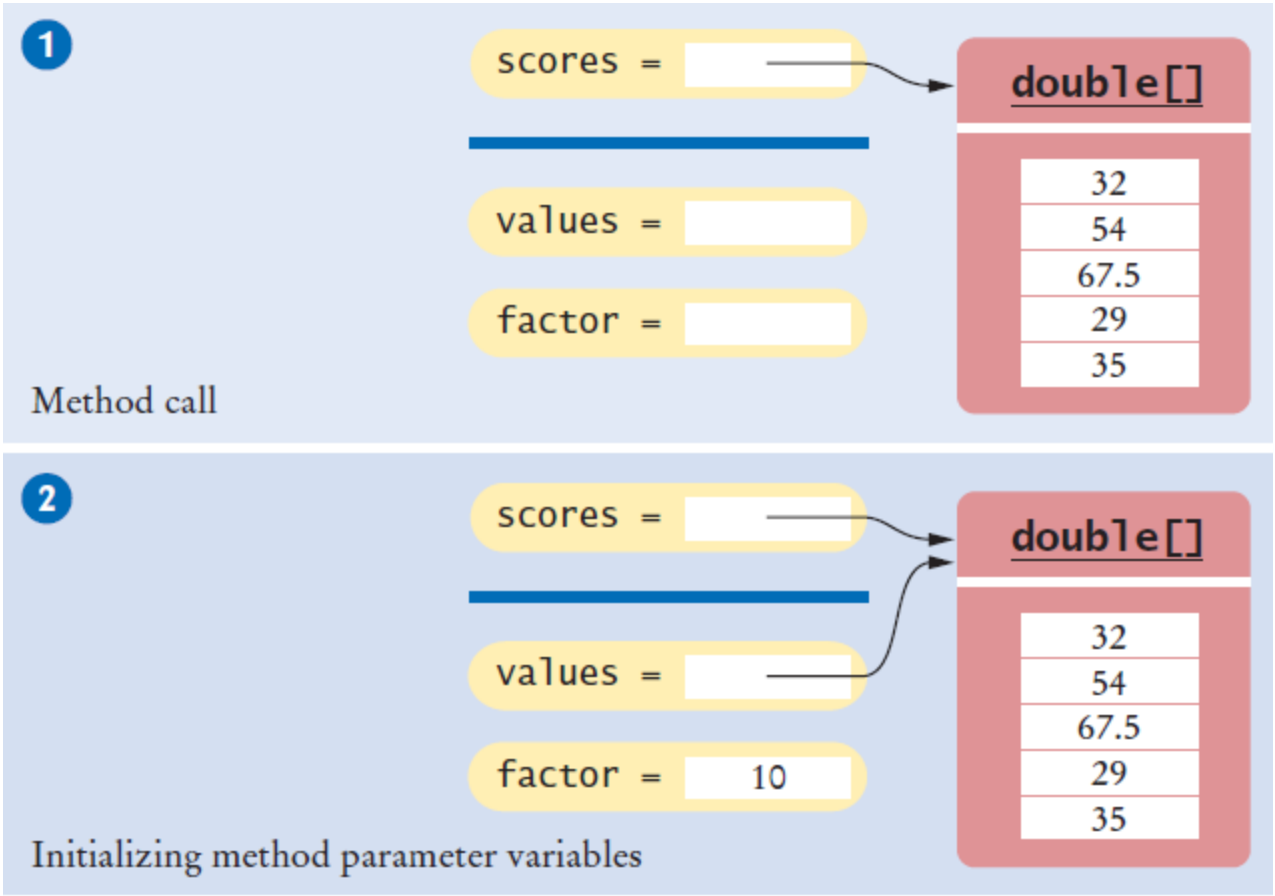
```
double[] scores = {32, 54, 67.5, 29, 35};

multiply(scores,    10);
```

reference value

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

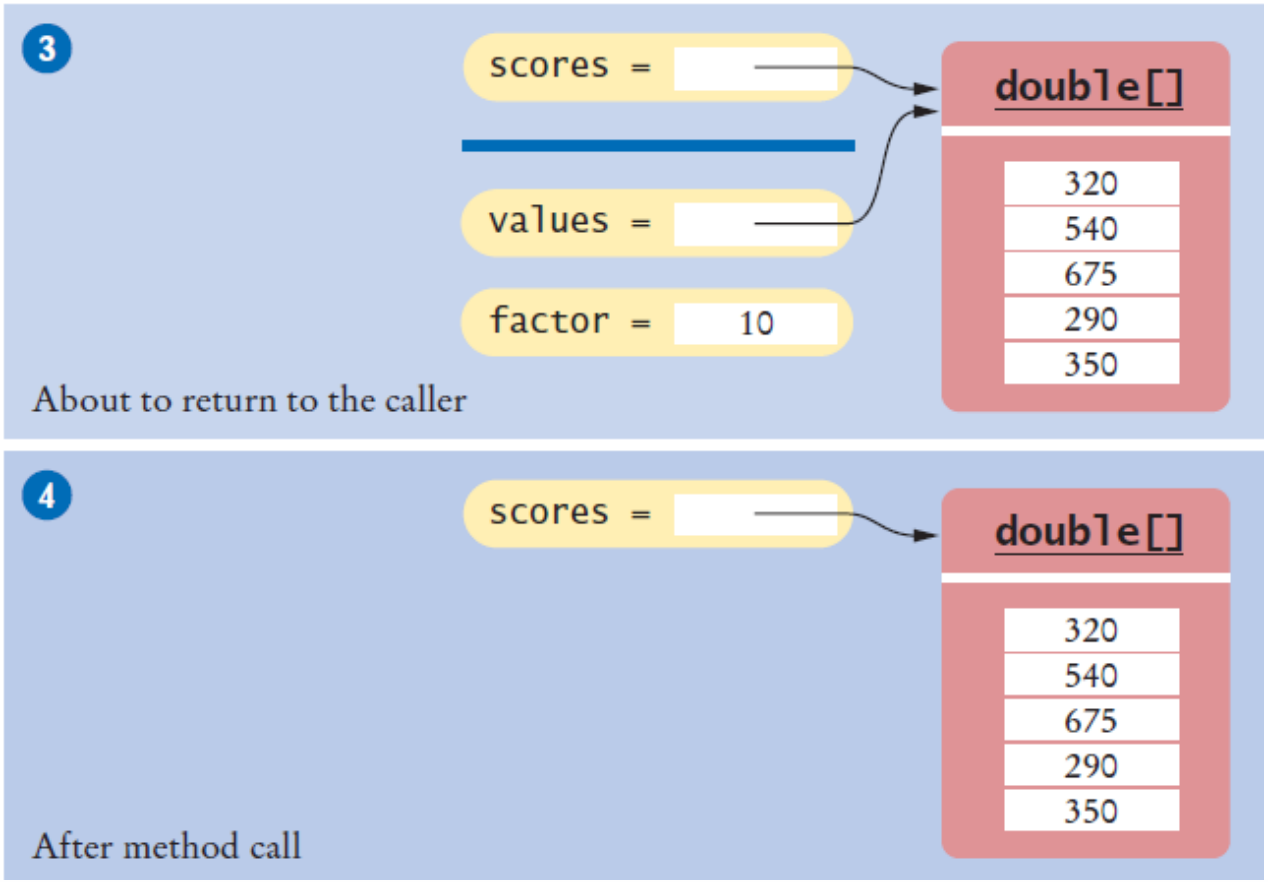
- The parameter variables `values` and `factor` are created. 1

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

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];
```

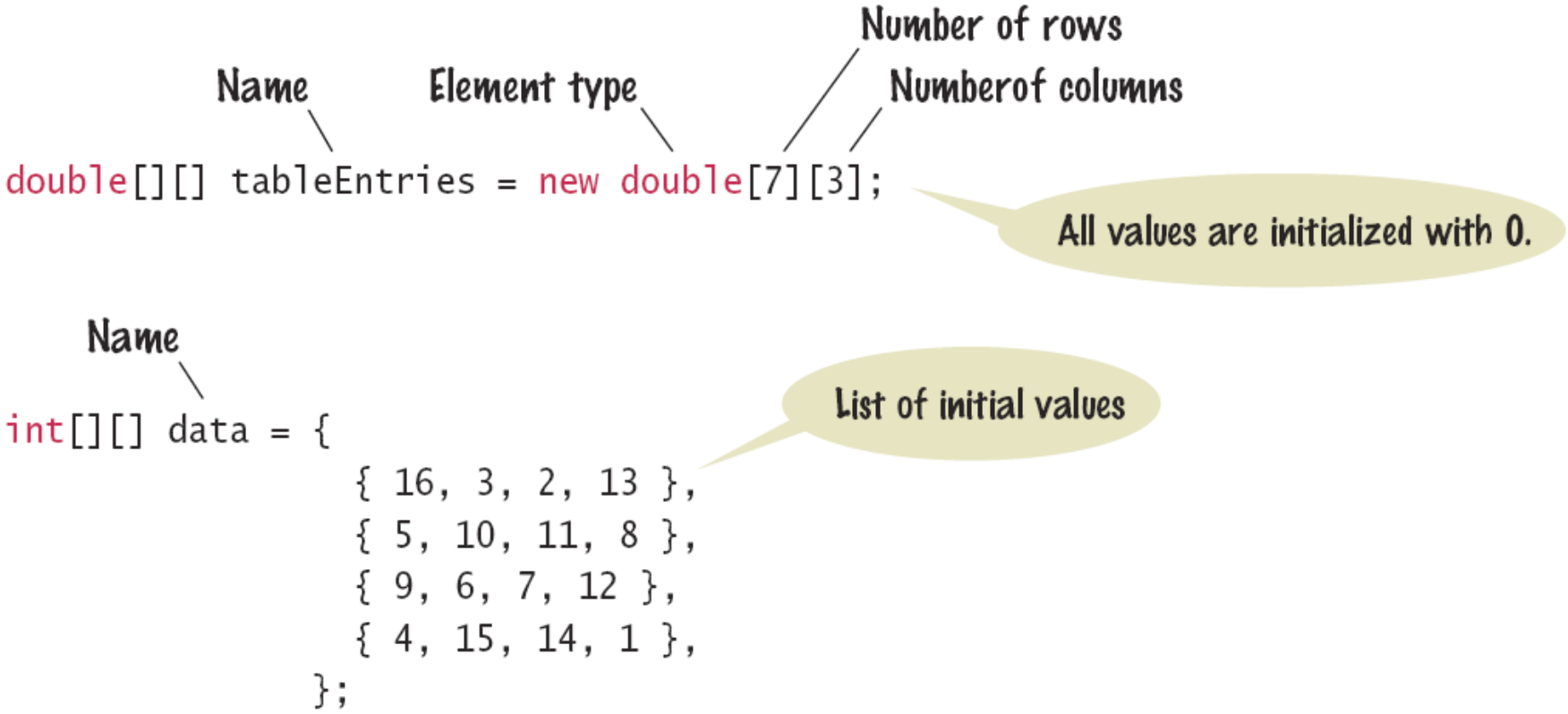
Gold	Silver	Bronze
1	0	1
1	1	0
0	0	1
1	0	0
0	1	1
0	1	1
1	1	0

- 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



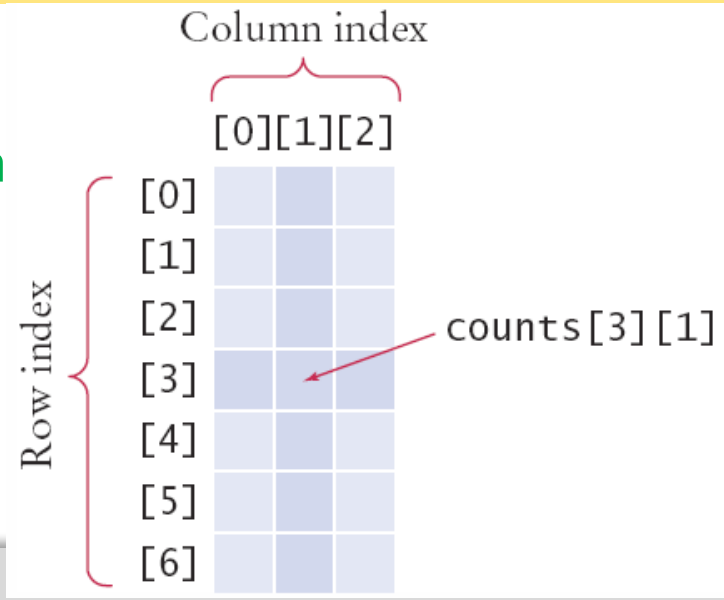
- The name of the array continues to be a reference to the contents of the array
  - Use new or fully initialize the array

Accessing Elements

- Use two index values:

Row then Column

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

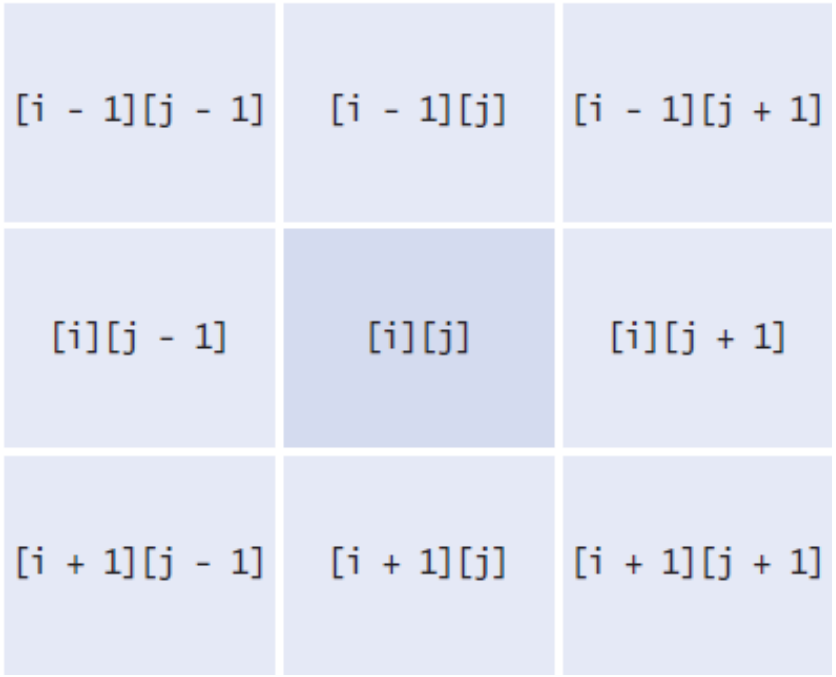


- To print
  - Use nested for loops
  - Outer row(i) , inner column(j) :

```
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

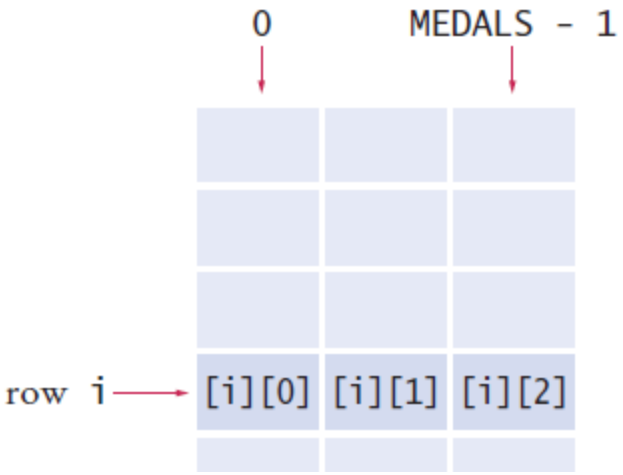
- Some programs that work with two-dimensional arrays need to locate the elements that are adjacent to an element
- This task is particularly common in games
- You are at loc `i, j`
- Watch out for edges!
  - No negative indexes!
  - Not off the 'board'



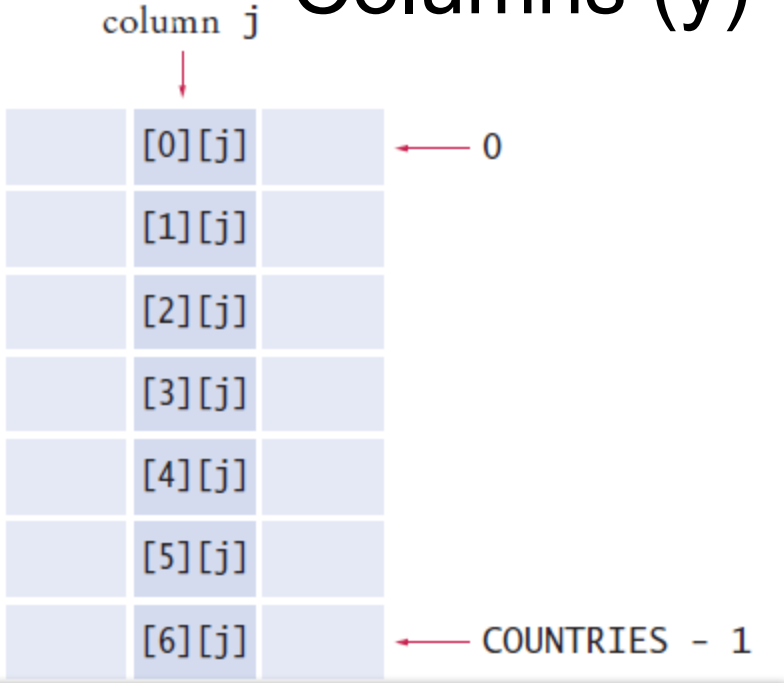
Adding Rows and Columns

- 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 n
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.

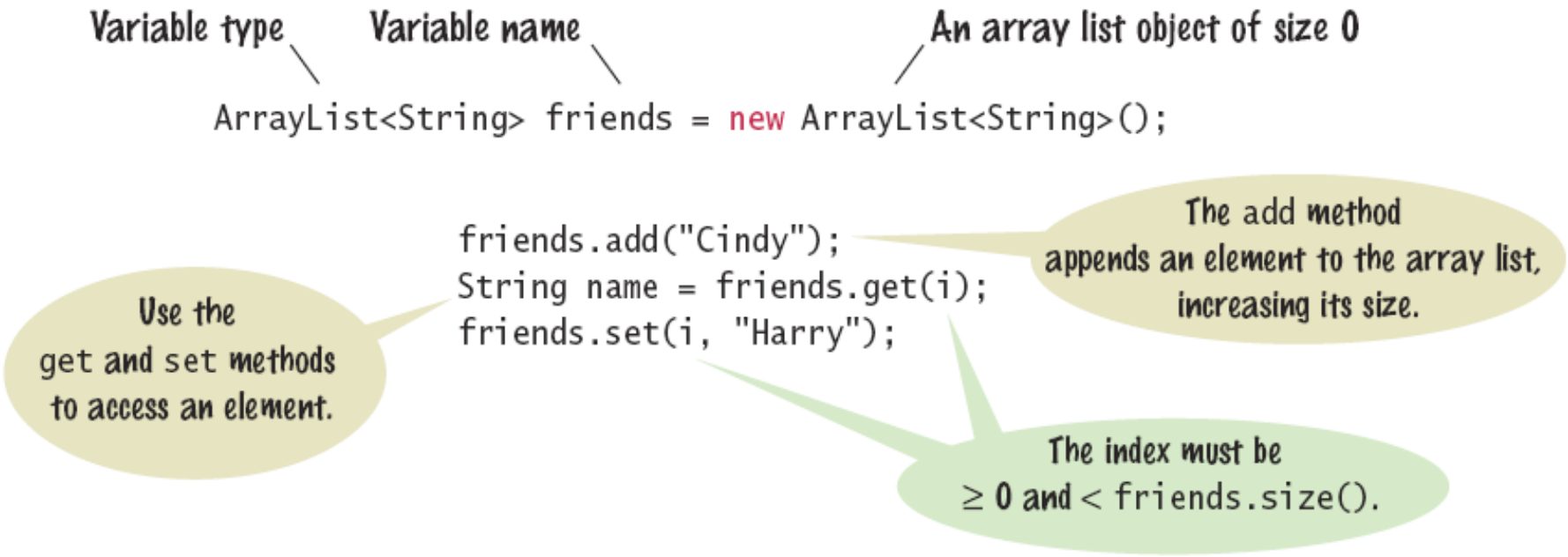
An Array List expands to hold as many elements as needed

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...`)

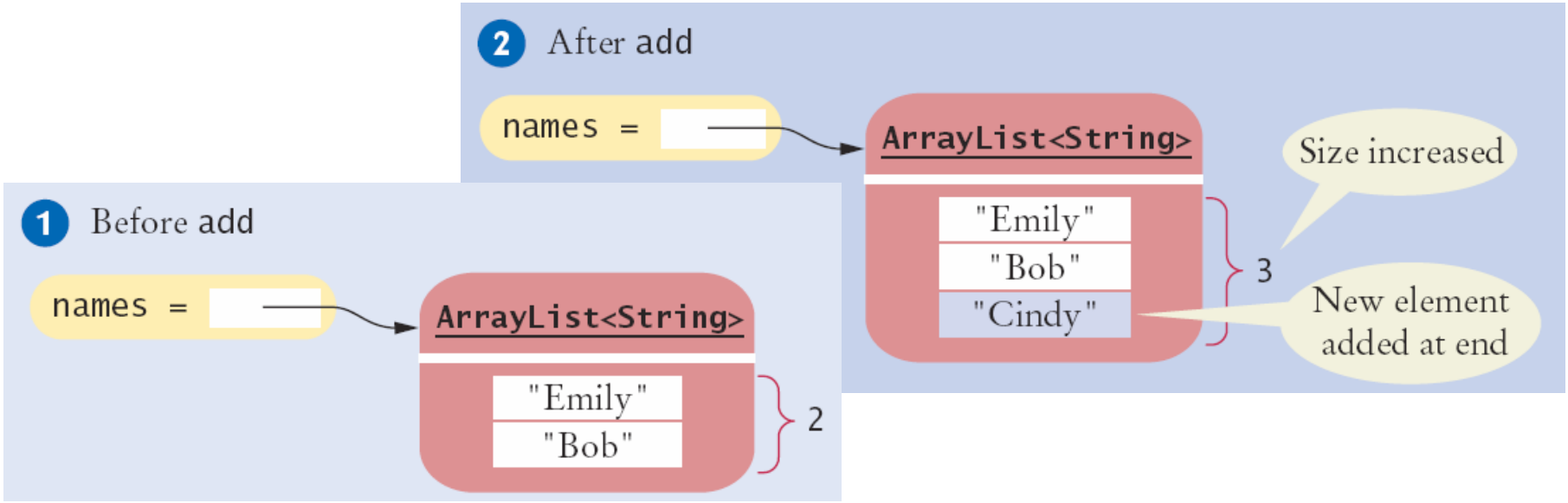
```
ArrayList<String> names = new ArrayList<String>();
```

Syntax 6.4: Array Lists



- `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

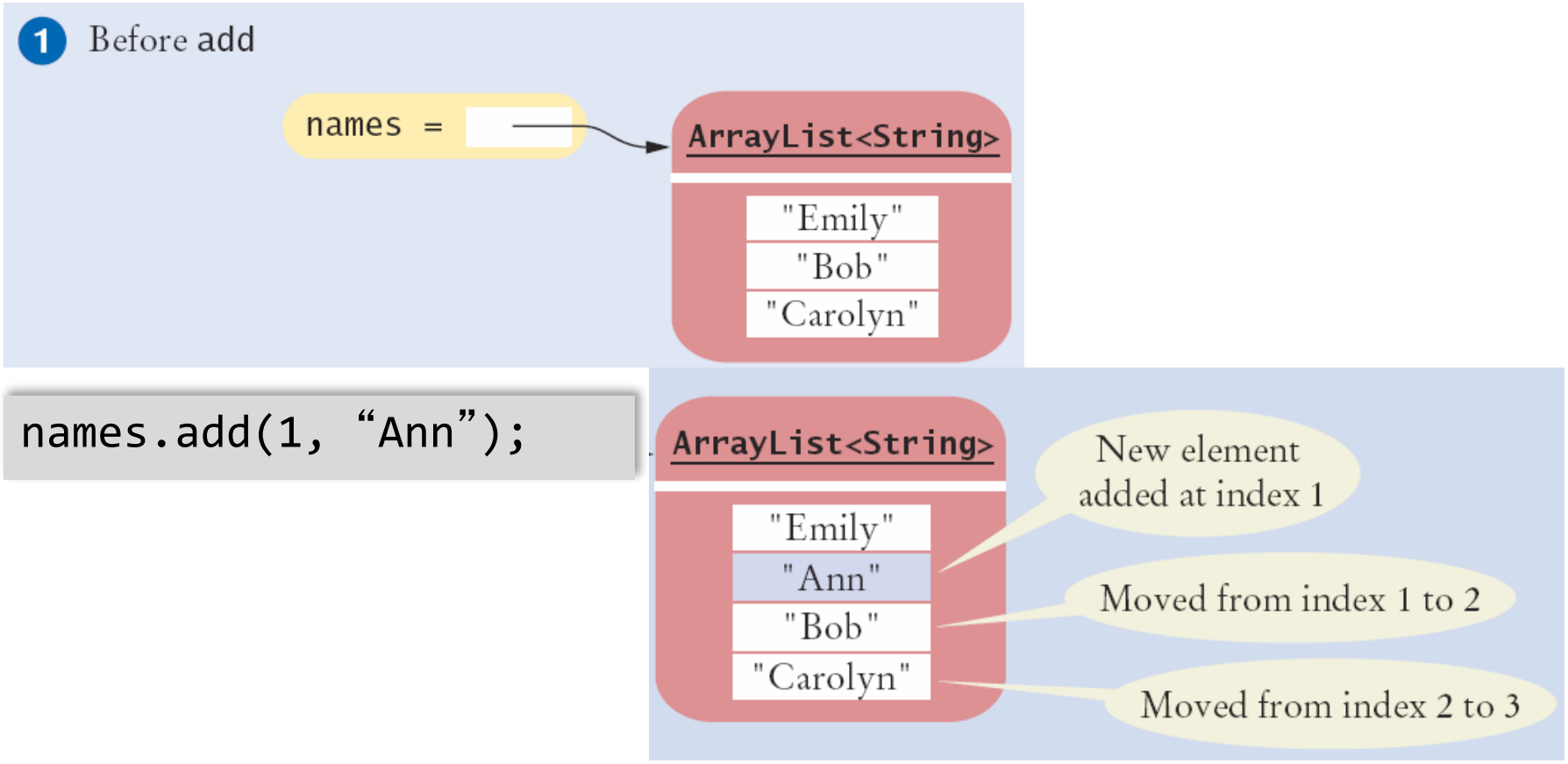
```
names.add("Cindy");
```

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

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

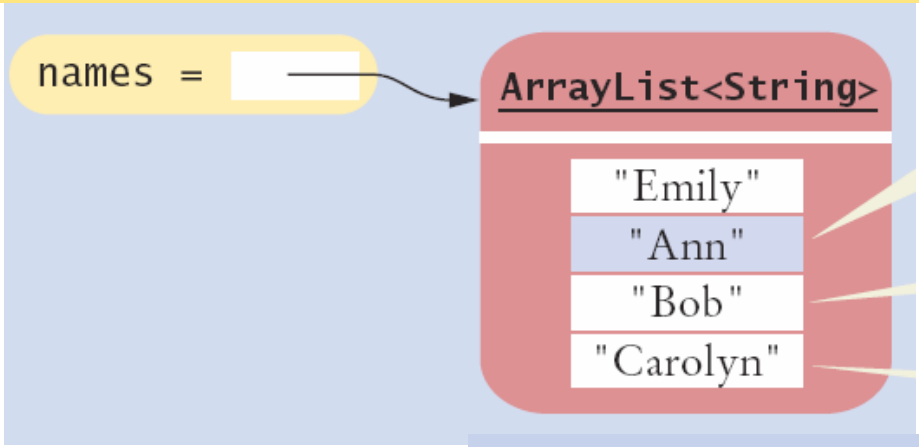
Moves all other elements

Adding an Element in the Middle

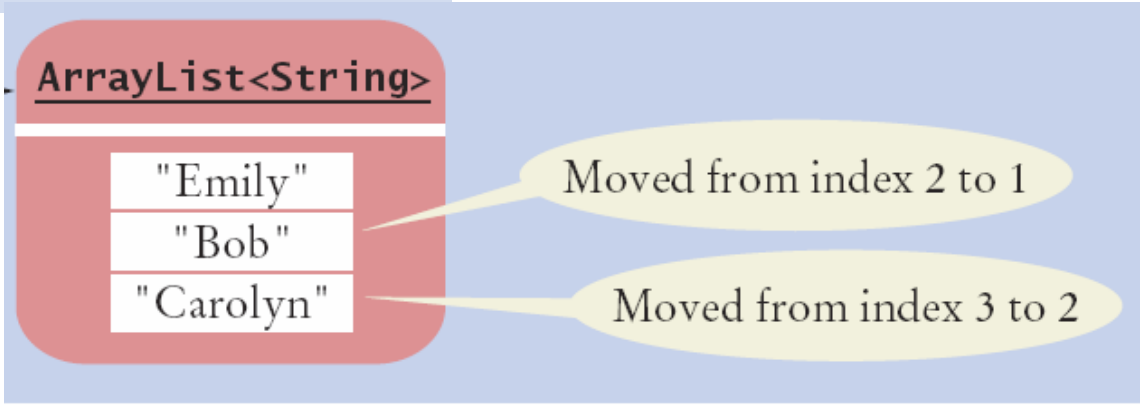


- Pass a location (index) and the new value to add
  Moves all other elements

Removing an Element




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



- 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);
}
```

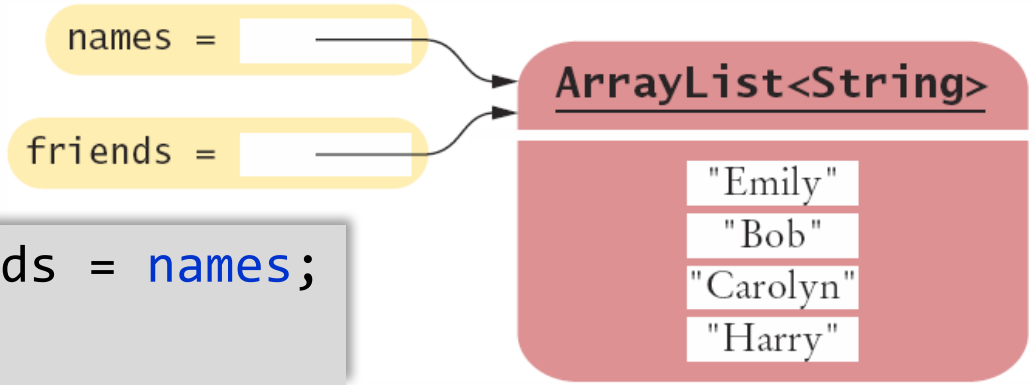
Working with Array Lists

Table 2 Working with Array Lists

`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 `Strings` and returns the reversed list.

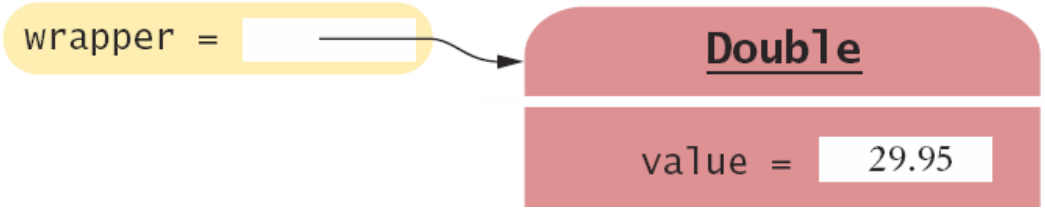
reference

```
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 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.

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

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: 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.