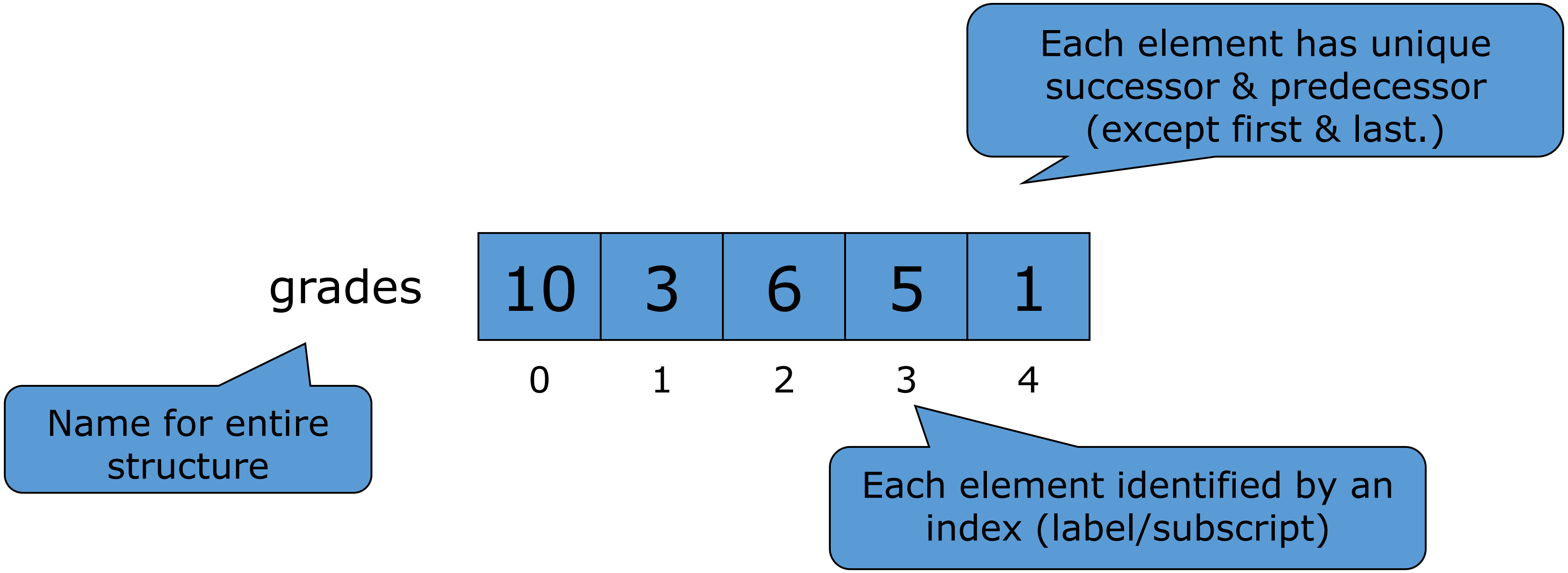
Java Coding 6

Collections

Not-so-easy Collections

- Arrays
  - Common data structure
  - All elements of same type
  - Are Objects in Java
  - Basis of ArrayList class!



Each element has unique successor & predecessor (except first & last.)

grades	10	3	6	5	1
	0	1	2	3	4

Name for entire structure

Each element identified by an index (label/subscript)

Array Syntax (1)

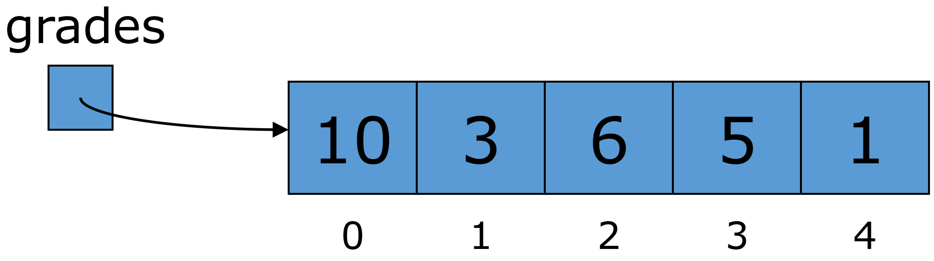
- Arrays are Objects, so
  - declare variable then instantiate

Array size cannot be changed after creation!

Note use of square brackets!

```
type[] variableName ;
```

```
variableName = new type[ noOfElements ];
```



grades

10	3	6	5	1
0	1	2	3	4

```
int[] grades;
grades = new int[5];
```

Array Syntax (1)

Initializer list

```
int[]  grades = {10, 3, 6, 5, 1};
```

Can only use this when declaring array, not afterwards!

Useful for constants such as

```
String[]  daysOfWeek = { “Mon”, “Tue”, “Wed”,
“Thu”, “Fri”, “Sat”, “Sun”};
```

Note: index must be between 0 & noOfElements
in array – 1,
else ArrayIndexOutOfBoundsException!

Array Syntax (2)

- Referring to an individual element

variableName[index]

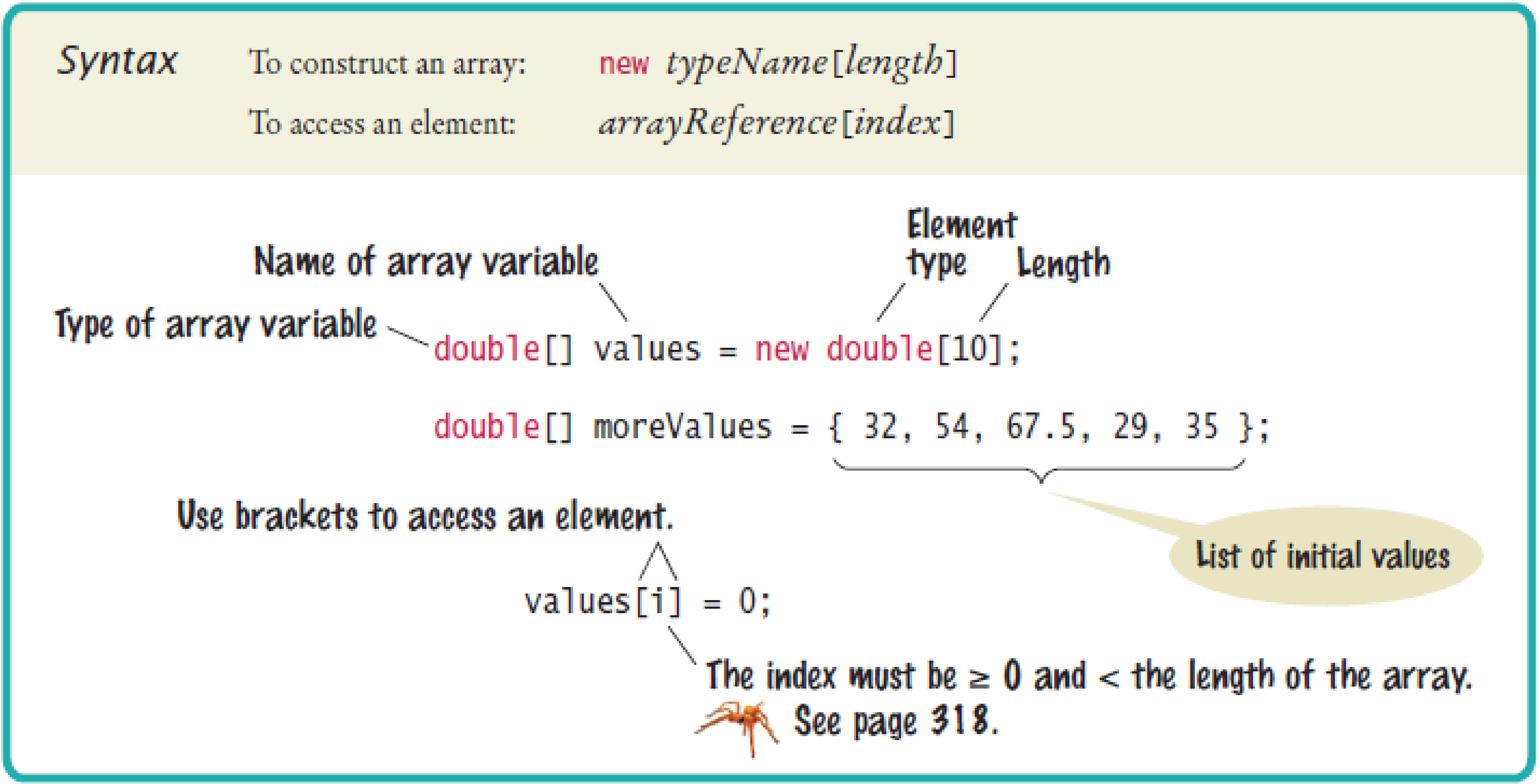
Where **index** is a literal, named constant, variable, or expression.

- examples

```
grades[0]    grades[ i]
grades[1]    grades[ i+1]
names[99]    names[ FIRST]
```

```
grades[0] = 10;
grades[1] = grades[0] + 2;
System.out.println( grades[0]);
names[99] = scan.nextLine();
```

Syntax 6.1 Arrays



Syntax

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

To access an element: *arrayReference*[*index*]

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

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

values[i] = 0;
```

Arrays – Bounds Error

- A bounds error occurs if you supply an invalid array index.
- Causes your program to terminate with a run-time error.
- Example:

```
double[] values = new double[10];
values[10] = value; // Error
```

- `values.length` yields the length of the `values` array.
- There are no parentheses following `length`.

Declaring Arrays

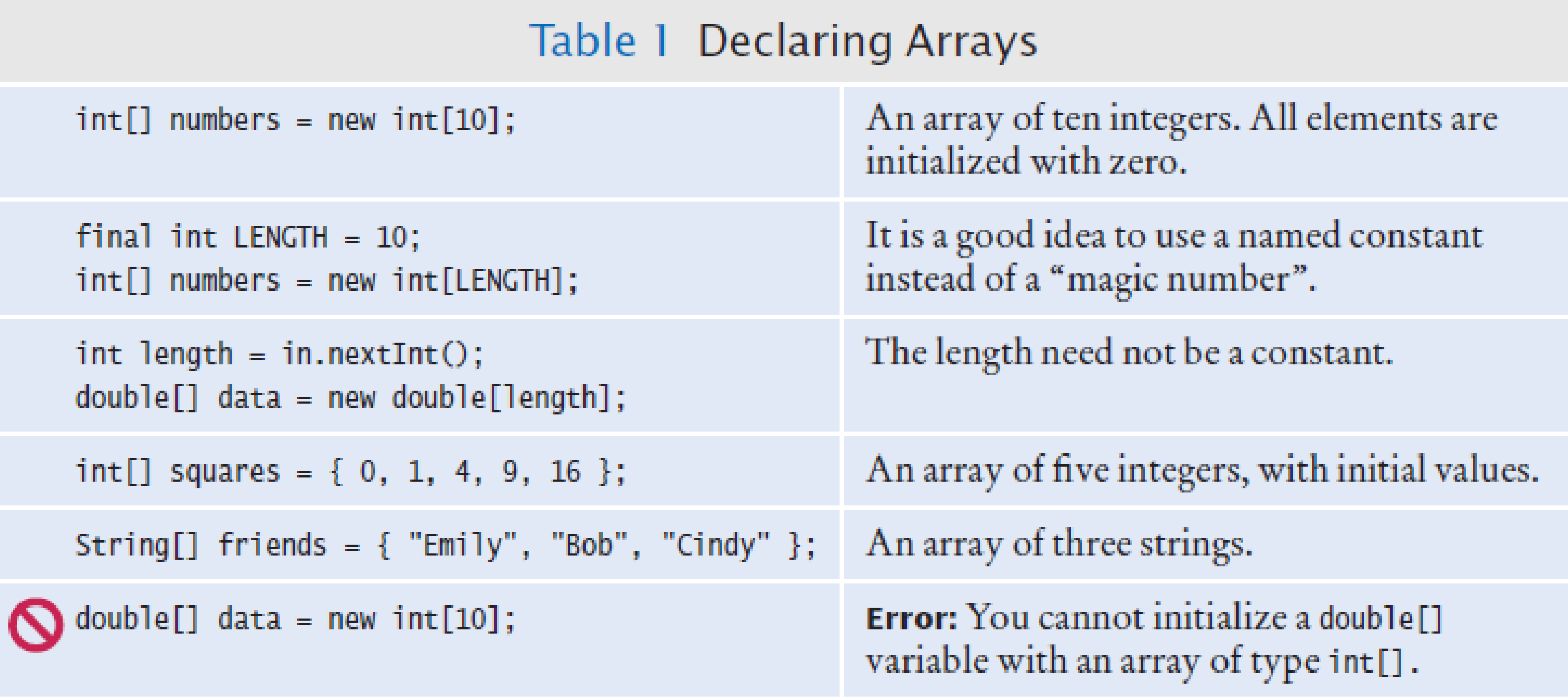


Table 1 Declaring Arrays

`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

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

Using Arrays with Methods

- Arrays can occur as method arguments and return values.
- An array as a method argument

```
public void addScores(int[] values)
{
   for (int i = 0; i < values.length; i++)
   {
      totalScore = totalScore + values[i];
   }
}
```

- To call this method

```
int[] scores = { 10, 9, 7, 10 };
fred.addScores(scores);
```

- A method with an array return value

```
public int[] getScores()
```

Cannot print array directly
System.out.println(grades): // doesn't work!

Processing all elements

- e.g. Printing contents of array grades

```
System.out.println( grades[0] );
System.out.println( grades[1] );
 :
```

```
for ( int i = 0; i < ___________; i++)
      System.out.println( grades[i] );
```

```
for ( int i = 0; i < grades.length; i++)
      System.out.println( grades[i] );
```

```
for each int k in grades array
      print k
```

```
// alternate for syntax
for ( int k : grades)
     System.out.println( k);
```

length is property (not method!) of arrays – returns
number of elements the array has.

The Enhanced `for` Loop

- You can use the enhanced for loop to visit all elements of an array.
- Totaling the elements in an array with the enhanced for loop

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

- The loop body is executed for each element in the array `values`.
- Read the loop as “for each element in `values`”.

The Enhanced for Loop

- Traditional alternative:

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

The Enhanced for Loop

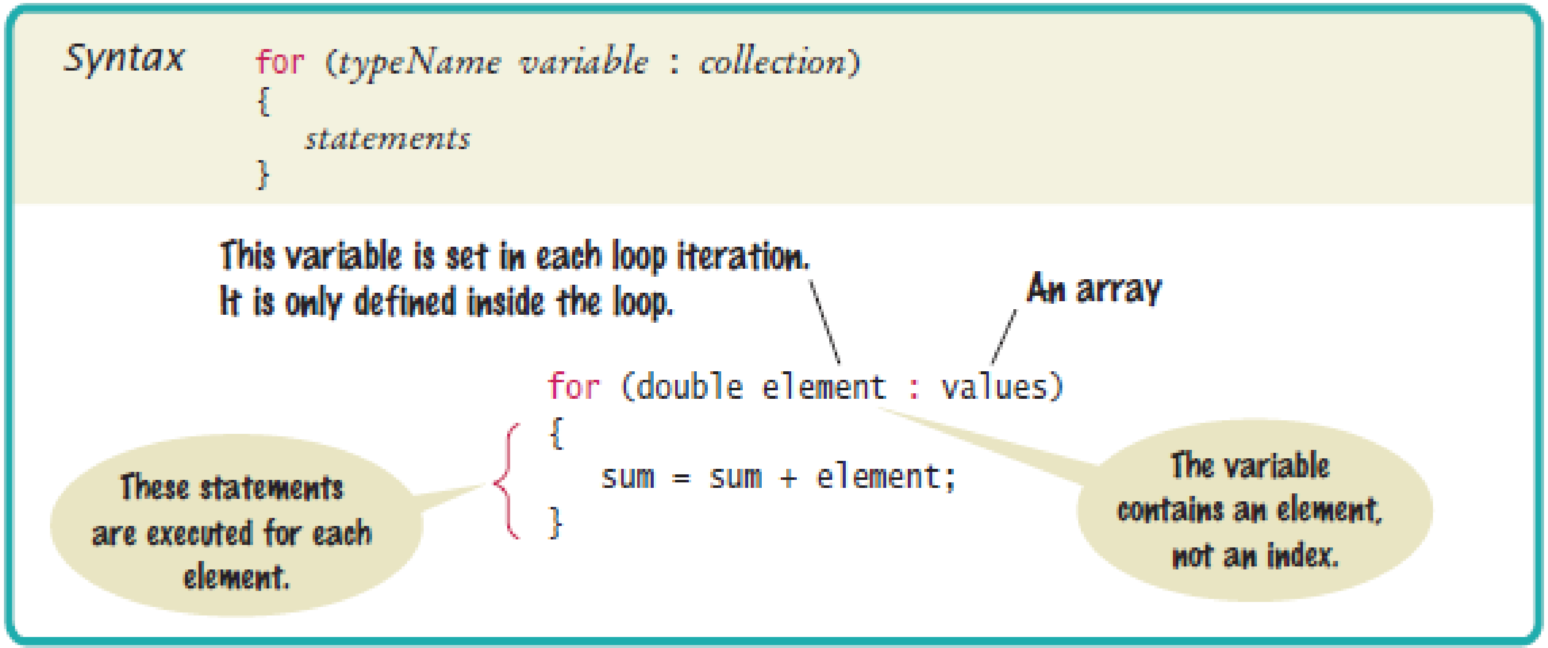
- Not suitable for all array algorithms.
- **Does not allow you to modify the contents of an array.**
- The following loop does not fill an array with zeros:

```
for (double element : values)
{
   element = 0;
   // ERROR: this assignment does not modify
   // array elements
}
```

- Use a basic for loop instead:

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

Syntax 6.2 The Enhanced for Loop



Syntax

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

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

An array

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

These statements are executed for each element.

The variable contains an element, not an index.

ArrayPlay - code

Easy Problem using arrays!

- Printing table of differences from average

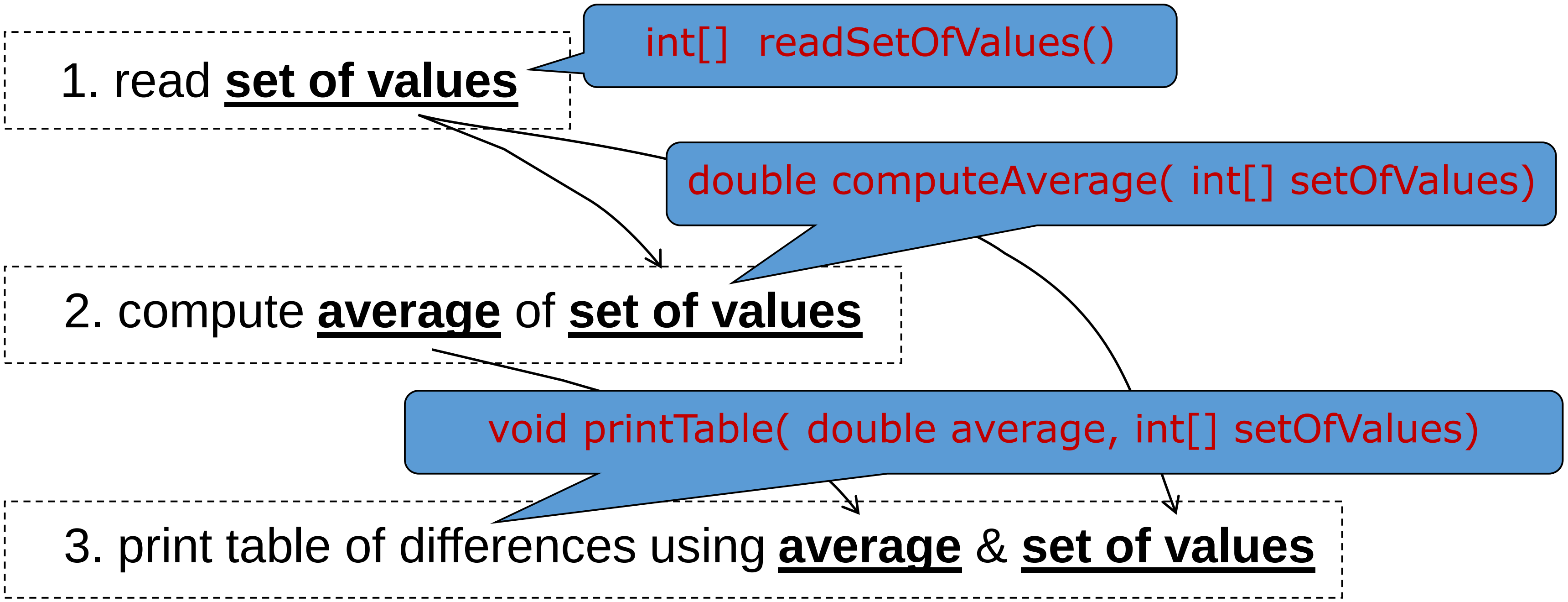
  1. read set of values
  2. compute average of set of values
  3. print table of differences using average & set of values

- Steps 2 & 3 are straightforward
- For step 1 need to know how many values
  - Fixed, e.g. 5
  - Ask user
  - Use sentinel - *but length of array is fixed!*

Easy Problem using arrays - code

Easy Problem with Methods!

- Identify method signatures from algorithm



1. read **set of values**

int[] readSetOfValues()

double computeAverage(int[] setOfValues)

2. compute **average** of **set of values**

void printTable(double average, int[] setOfValues)

3. print table of differences using **average** & **set of values**

Data Requirements:
average – double
setOfValues – int[]

Note: Object-type parameters
can act as outputs too!

Common Array Algorithm: Filling

- Fill an array with squares (0, 1, 4, 9, 16, ...):

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

Common Array Algorithm: Maximum or Minimum

- Finding the maximum in an array

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

- The loop starts at 1 because we initialize `largest` with `values[0]`.

© CEFutcher/iStockphoto.

Common Array Algorithm: Linear Search

- To find the position of an element:
  - Visit all elements until you have found a match or you have come to the end of the array
- Example: Find the first element that is equal to 100

```
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"); }
```

Common Array Algorithm: Removing an Element

Problem: To remove the element with index pos from the array values with number of elements currentSize.

- Unordered
  1. Overwrite the element to be removed with the last element of the array.
  2. Decrement the currentSize variable.

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

Common Array Algorithm: Removing an Element

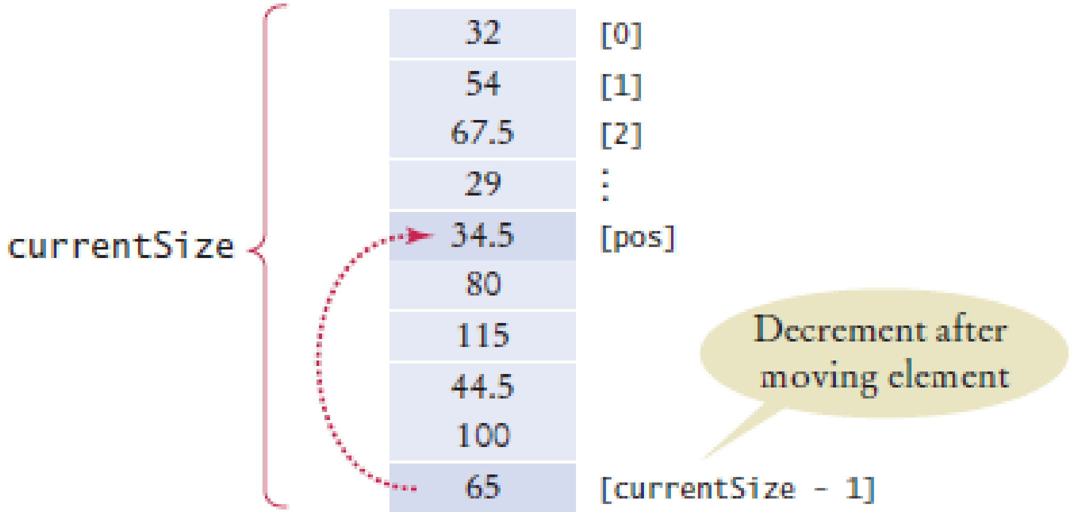


Figure 6 Removing an Element in an Unordered Array

Common Array Algorithm: Removing an Element

- Ordered array
  1. Move all elements following the element to be removed to a lower index.
  2. Decrement the variable holding the size of the array.

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

Common Array Algorithm: Removing an Element

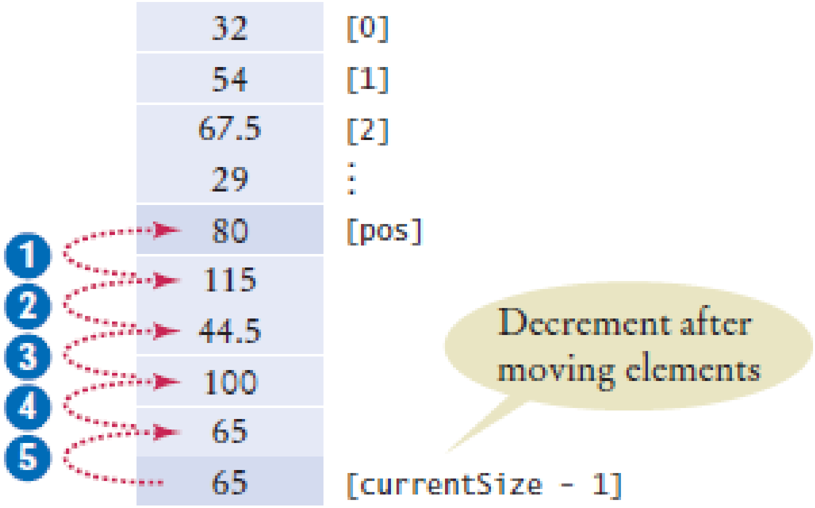


Figure 7 Removing an Element in an Ordered Array

Common Array Algorithm: Inserting an Element

- If order does not matter
  1. Insert the new element at the end of the array.
  2. Increment the variable tracking the size of the array.

```
if (currentSize < values.length)
{
   currentSize++;
   values[currentSize -1 ] = newElement;
}
```

Common Array Algorithm: Inserting an Element

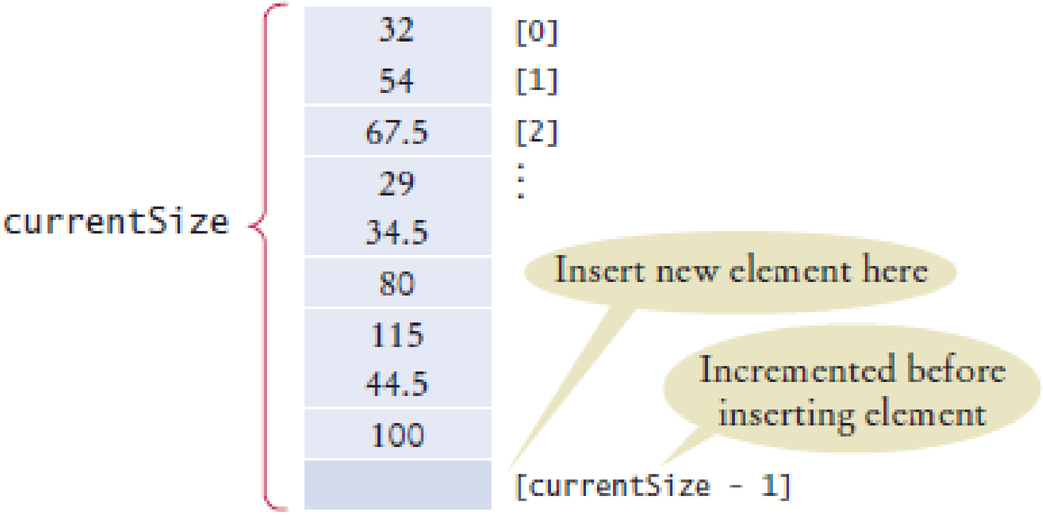


Figure 8 Inserting an Element in an Unordered Array

Common Array Algorithm: Inserting an Element

- If order matters Increment the variable tracking the size of the array.
  1. Move all elements after the insertion location to a higher index.
  2. Insert the element.

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

Common Array Algorithm: Inserting an Element

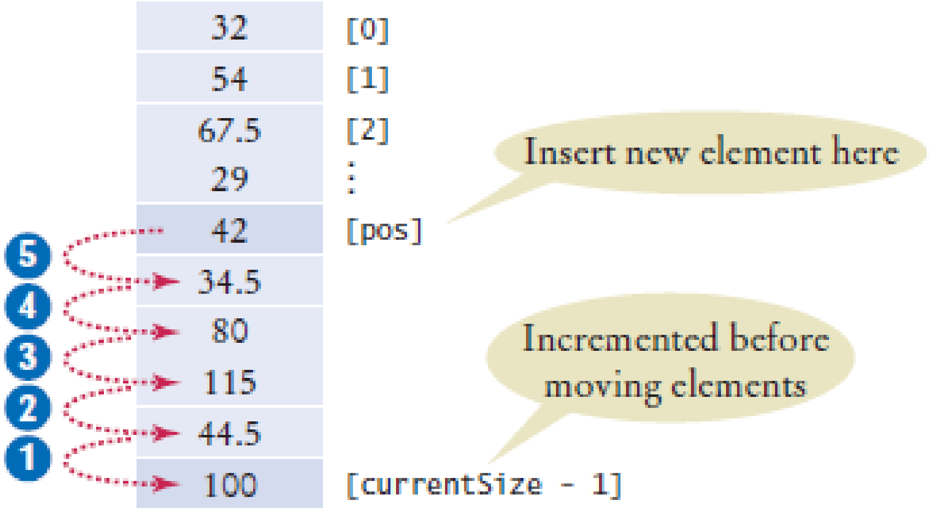
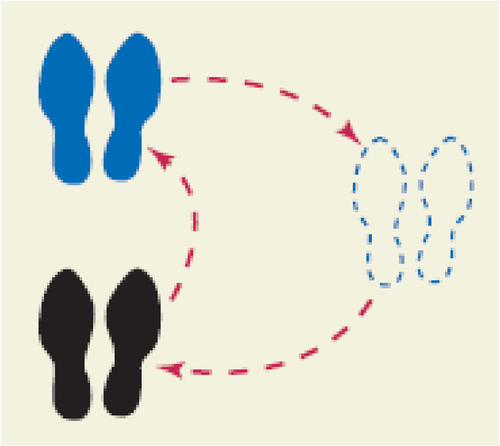


Figure 9 Inserting an Element in an Ordered Array

Common Array Algorithm: Swapping Elements

- To swap two elements, you need a temporary variable.



- We need to save the first value in the temporary variable before replacing it.

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

- Now we can set `values[j]` to the saved value.

```
values[j] = temp;
```

Common Array Algorithm: Swapping Elements

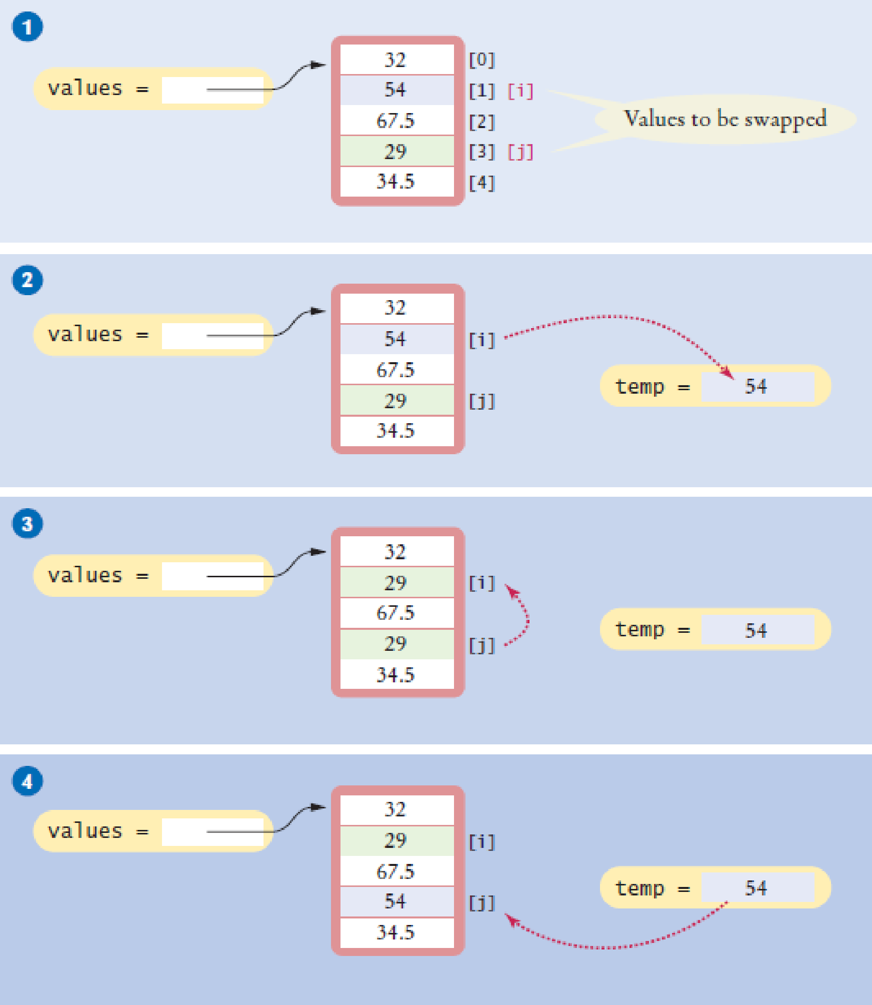


Figure 10 Swapping Array Elements

Common Array Algorithm: Copying an Array

- Copying an array variable yields a second reference to the same array:

```
double[] values = new double[6];
. . . // Fill array
double[] prices = values; ❶
```

- To make a true copy of an array, call the `Arrays.copyOf` method:

```
double[] prices =
   Arrays.copyOf(values, values.length); ❷
```

Common Array Algorithm: Copying an Array

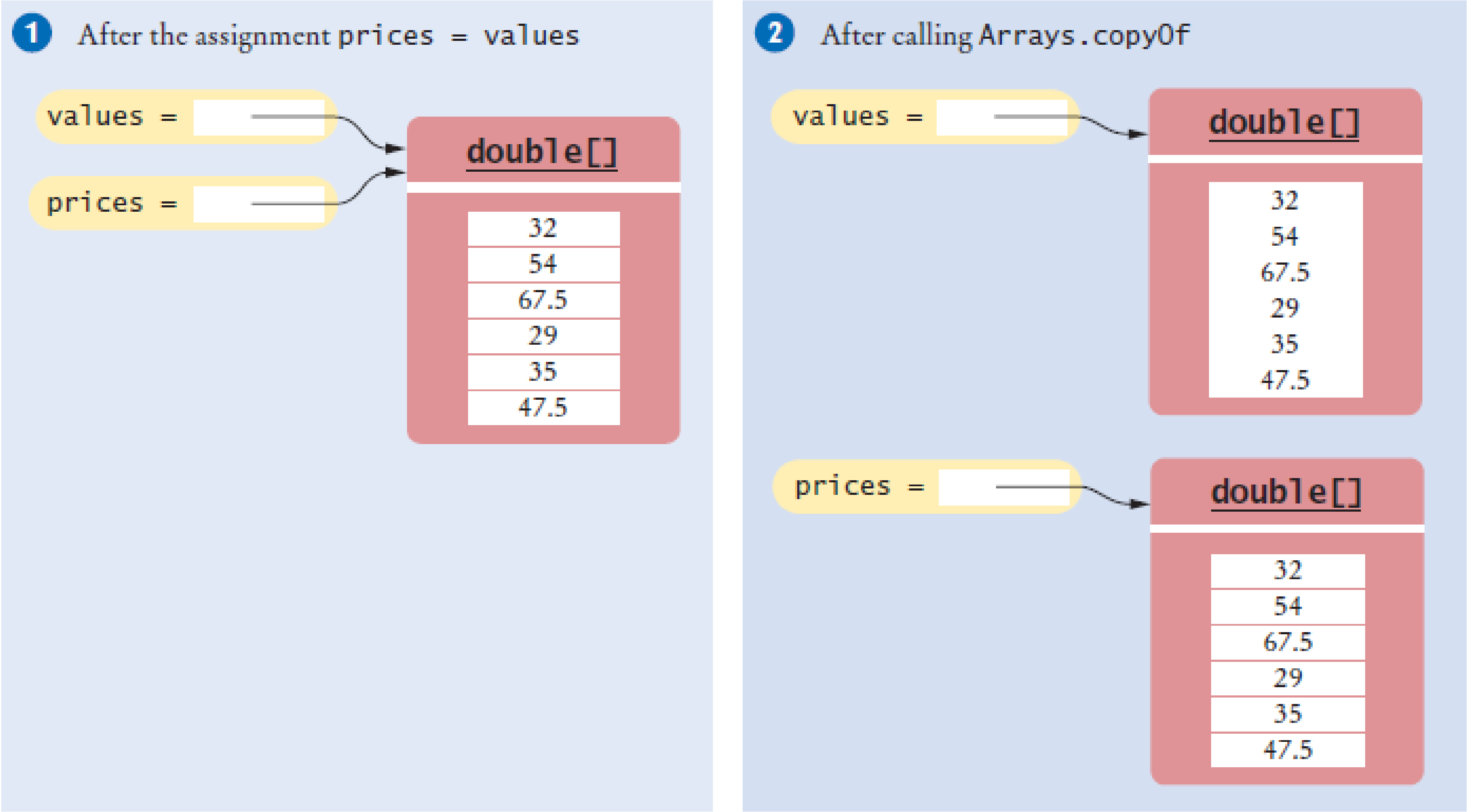


Figure 11 Copying an Array Reference versus Copying an Array

Common Array Algorithm: Growing an Array

- To grow an array that has run out of space, use the Arrays.copyOf method to double the length of an array

```
double[] newValues = Arrays.copyOf(values, 2 * values.length); ❶
values = newValues; ❷
```

Common Array Algorithm: Growing an Array

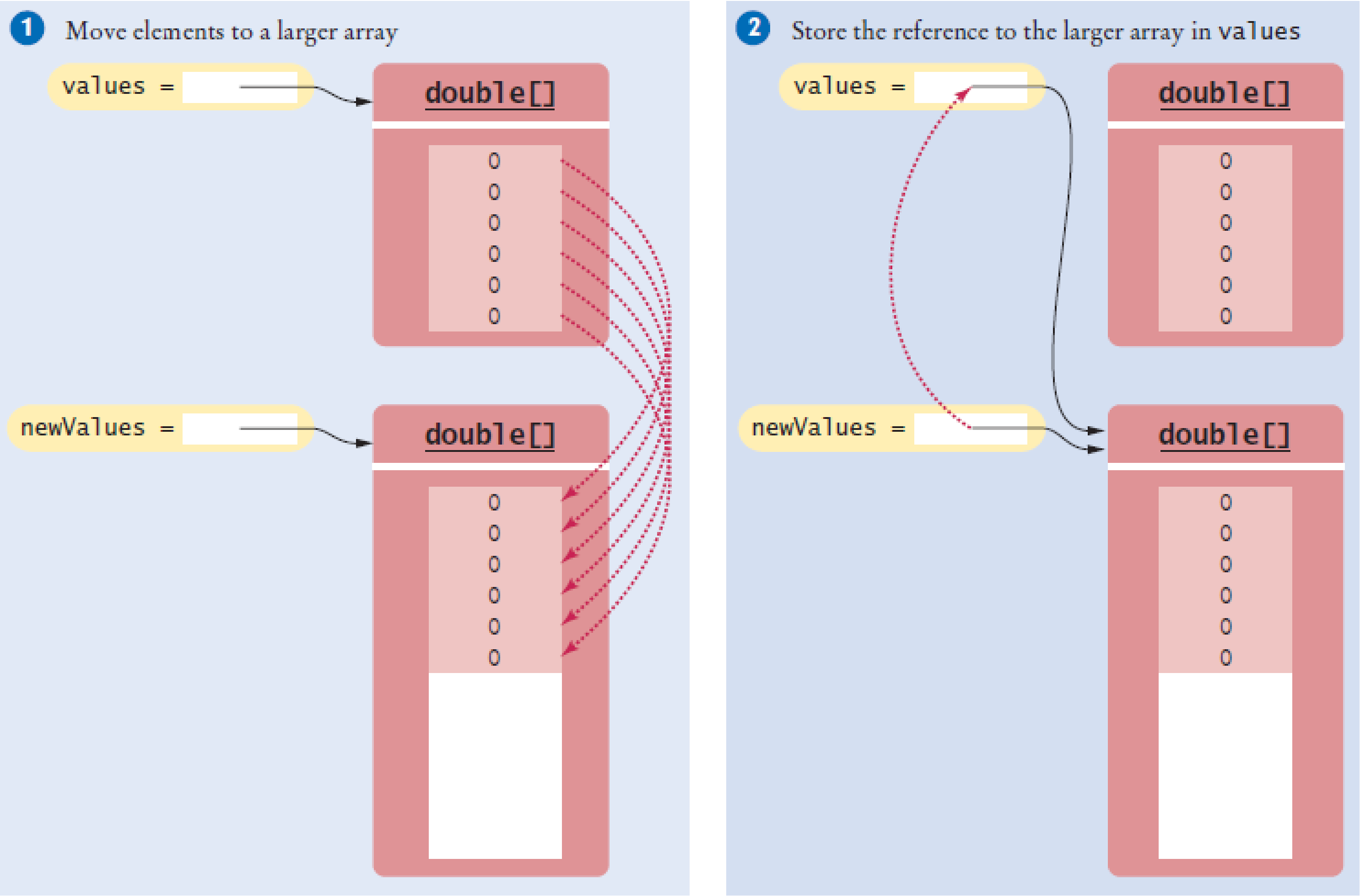


Figure 12 Growing an Array

Reading Input

- To read a sequence of arbitrary length:
  - Add the inputs to an array until the end of the input has been reached.
  - Grow when needed.

```
double[] inputs = new double[INITIAL_SIZE];
int currentSize = 0;
while (in.hasNextDouble())
{
   // Grow the array if it has been completely filled
   if (currentSize >= inputs.length)
   {
   inputs = Arrays.copyOf(inputs, 2 * inputs.length); // Grow the inputs array
   }
   inputs[currentSize] = in.nextDouble(); currentSize++;
}
```

  - Discard unfilled elements.

```
inputs = Arrays.copyOf(inputs, currentSize);
```

section_3/LargestInArray.java

This program reads a sequence of values and prints them, marking the largest value.

Program Run

```
Please enter values, Q to quit: 34.5 80 115 44.5 Q
                       34.5
                        80
              115 <== largest value
                       44.5
```

section_3/LargestInArray.java

```java
import java.util.Scanner;

/**
   This program reads a sequence of values and prints them, marking the largest value.
*/
public class LargestInArray
{
   public static void main(String[] args)
   {
      final int LENGTH = 100;
      double[] values = new double[LENGTH];
      int currentSize = 0;

      // Read inputs

      System.out.println("Please enter values, Q to quit:");
      Scanner in = new Scanner(System.in);
      while (in.hasNextDouble() && currentSize < values.length)
      {
         values[currentSize] = in.nextDouble();
         currentSize++;
      }

```

Continued

section_3/LargestInArray.java

```
      // Find the largest value

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

      // Print all values, marking the largest

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

***Continu
ed***

section_3/LargestInArray.java

Program Run

```
Please enter values, Q to quit: 34.5 80 115 44.5 Q
                       34.5
                        80
              115 <== largest value
                       44.5
```

Self Check 6.13

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 6.14

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 6.15

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

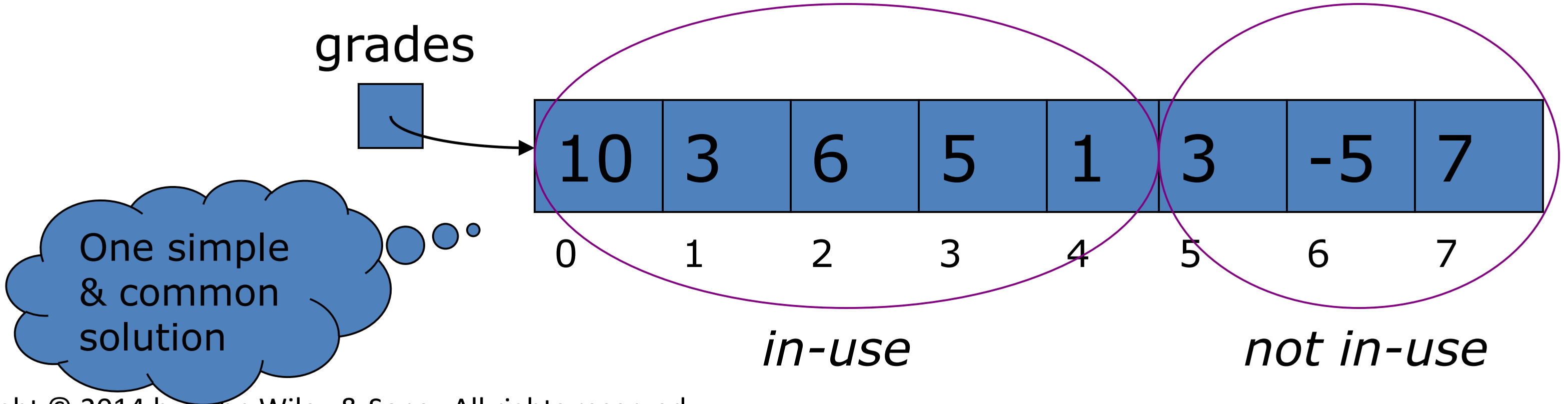
```
double largest = 0;
for (int i = 0; i < values.length; i++)
{
   if (values[i] > largest) { largest = values[i]; }
}
```

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

Using part of an array (1)

- Array size specified & fixed at instantiation
- Problem
  - if required size is unknown?
- Solution
  - make big enough for worst-case & use part of it

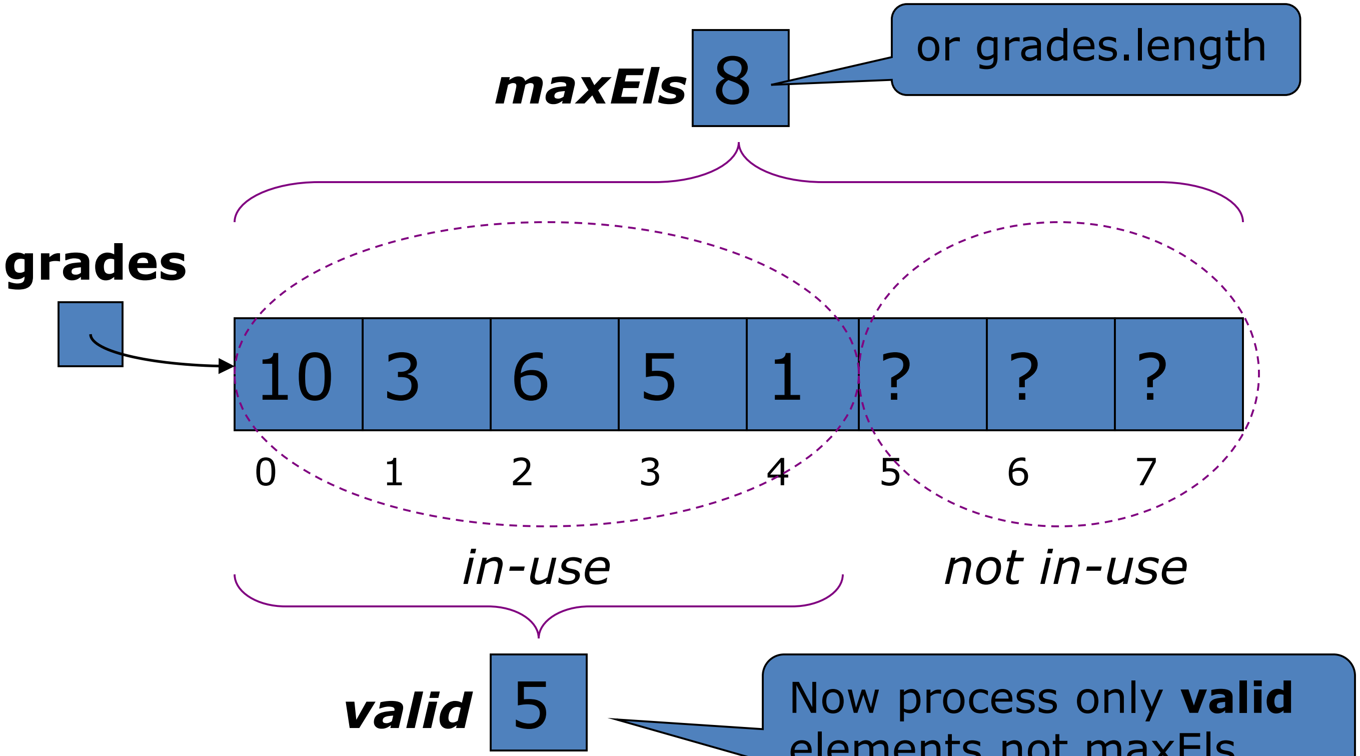
    Must divide array into two sets, in-use & not in-use … but how?



One simple & common solution

Using part of an array (2)

- Store elements sequentially from element zero
- Keep count of number of in-use elements (valid)



Now process only **valid** elements not maxEls

Partially Filled Arrays

- Array length = maximum number of elements in array.
- Usually, array is partially filled
- Define an array larger than you will need

```
final int LENGTH = 100;
double[] values = new double[LENGTH];
```

- Use companion variable to keep track of current size: call it `currentSize`

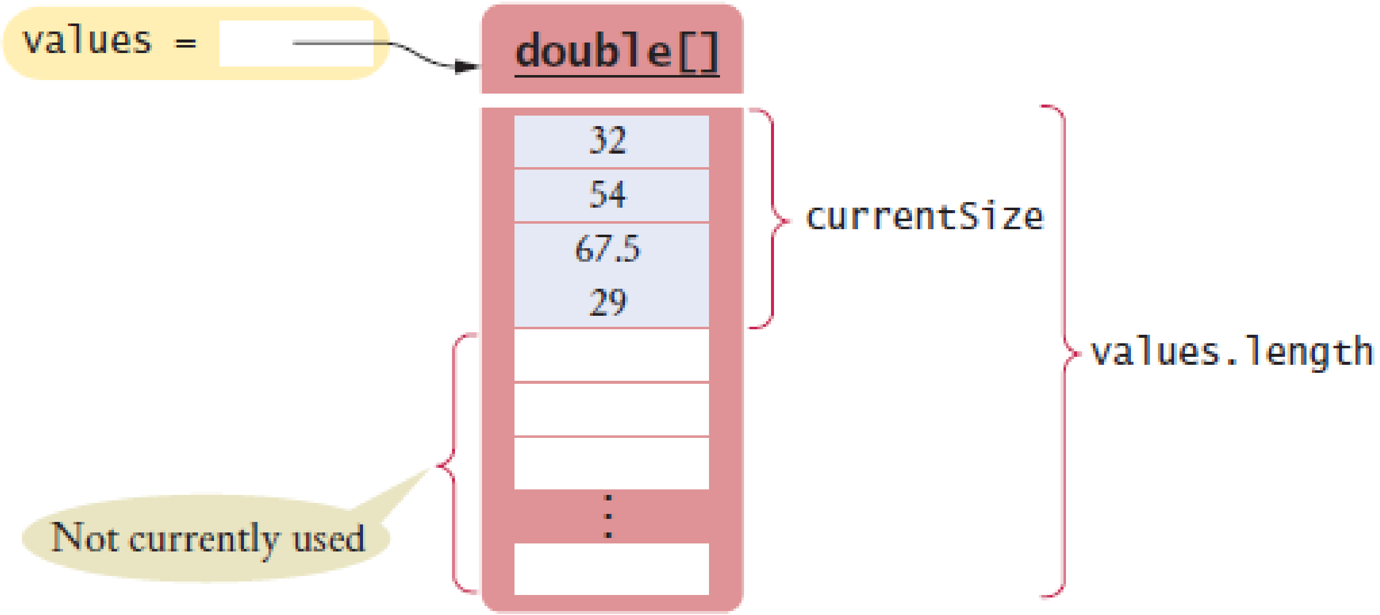
Partially Filled Arrays

- A loop to fill the array

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

- At the end of the loop, `currentSize` contains the actual number of elements in the array.
- Note: Stop accepting inputs when `currentSize` reaches the array length.

Partially Filled Arrays



Partially Filled Arrays

- To process the gathered array elements, use the companion variable, not the array length:

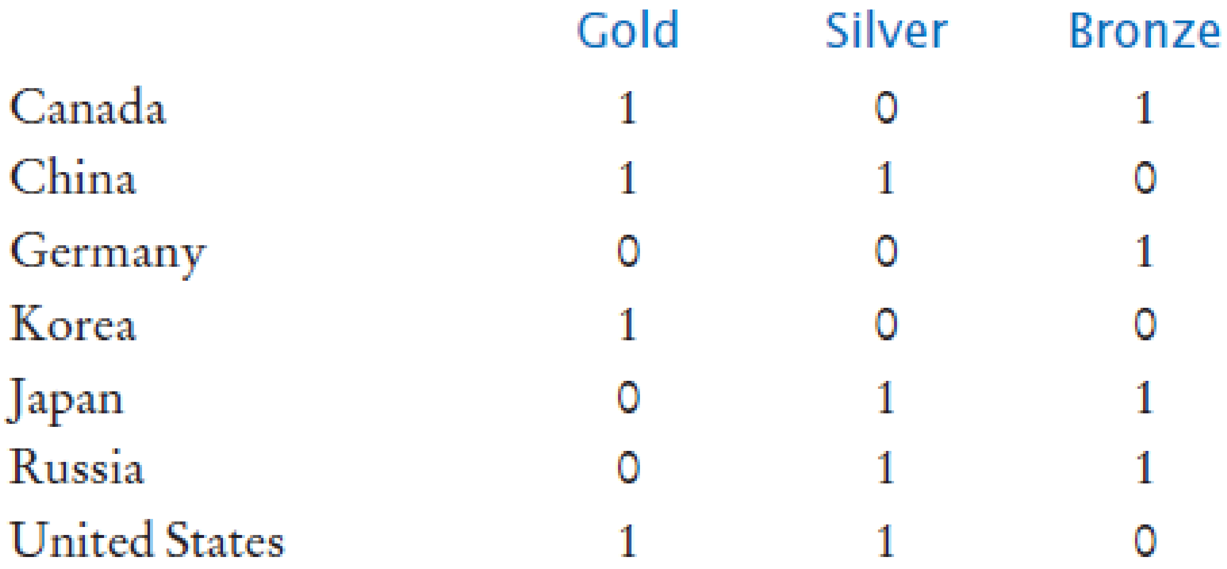
```
for (int i = 0; i < currentSize; i++)
{
   System.out.println(values[i]);
}
```

- With a partially filled array, you need to remember how many elements are filled.

© AlterYourReality/iStockphoto.

Two-Dimensional Arrays

- An arrangement consisting of rows and columns of values
  - Also called a matrix.
- Example: medal counts of the figure skating competitions at the 2010 Winter Olympics.



	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 13 Figure Skating Medal counts

Two-Dimensional Arrays

- Use a two-dimensional array to store tabular data.
- When constructing a two-dimensional array, specify how many rows and columns are needed:

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

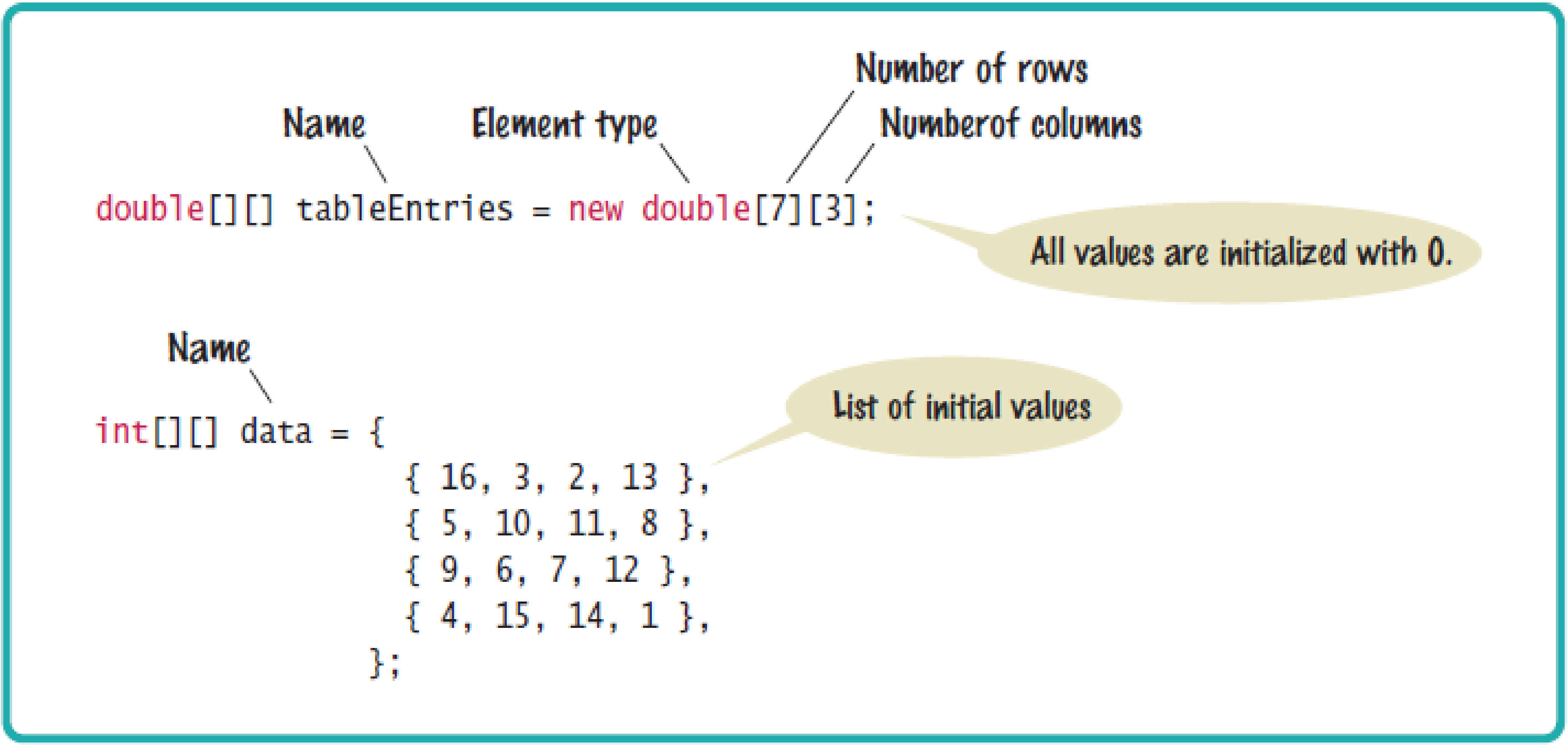
Two-Dimensional Arrays

- You can declare and initialize the array by grouping each row:

```
int[][] counts =
{
   { 1, 0, 1 },
   { 1, 1, 0 },
   { 0, 0, 1 },
   { 1, 0, 0 },
   { 0, 1, 1 },
   { 0, 1, 1 },
   { 1, 1, 0 }
};
```

- You cannot change the size of a two-dimensional array once it has been declared.

Syntax 6.3 Two-Dimensional Array Declaration



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

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

Accessing Elements

- Access by using two index values, `array[i][j]`

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

- Use nested loops to access all elements in a two-dimensional array.
- Example: print all the elements of the counts array

```
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
}
```

Accessing Elements

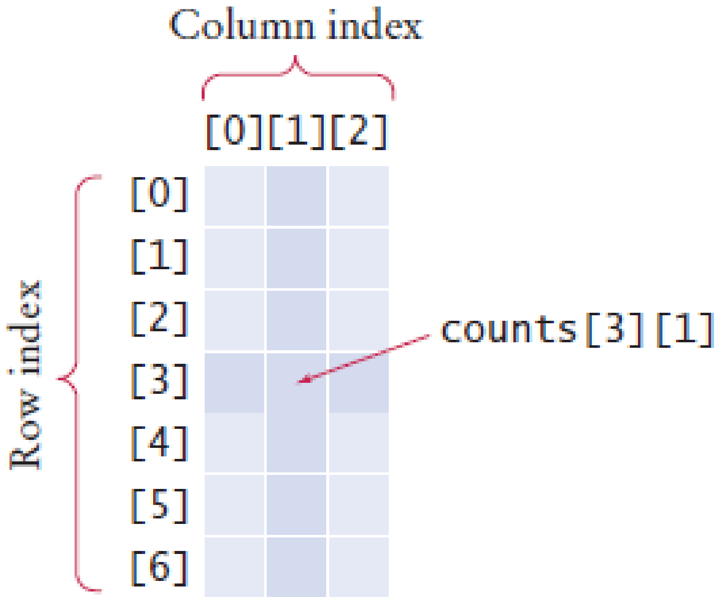


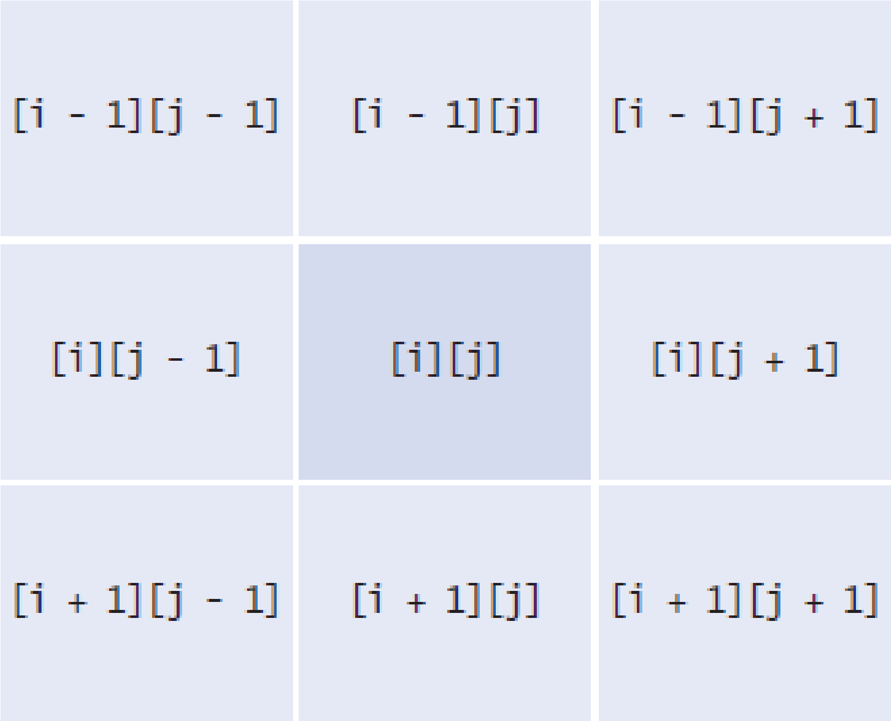
Figure 14 Accessing an Element in a Two-Dimensional Array

Accessing Elements

- Number of rows: `counts.length`
- Number of columns: `counts[0].length`
- Example: print all the elements of the `counts` array

```
for (int i = 0; i < counts.length; i++)
{
   for (int j = 0; j < counts[0].length; j++)
   {
      System.out.printf("%8d", counts[i][j]);
   }
   System.out.println();
}
```

Locating Neighboring Elements



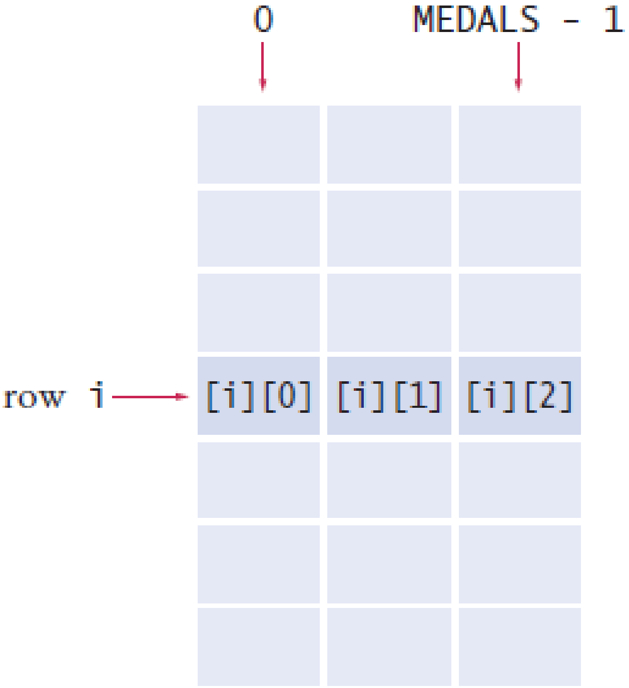
[i - 1][j - 1]	[i - 1][j]	[i - 1][j + 1]
[i][j - 1]	[i][j]	[i][j + 1]
[i + 1][j - 1]	[i + 1][j]	[i + 1][j + 1]

Figure 15 Neighboring Locations in a Two-Dimensional Array

- Watch out for elements at the boundary array
  - `counts[0][1]` does not have a neighbor to the top

Accessing Rows and Columns

- Problem: To find the number of medals won by a country
  - Find the sum of the elements in a row
- To find the sum of the `i`th row
  - compute the sum of `counts[i][j]`, where `j` ranges from `0` to `MEDALS - 1`.



Accessing Rows and Columns

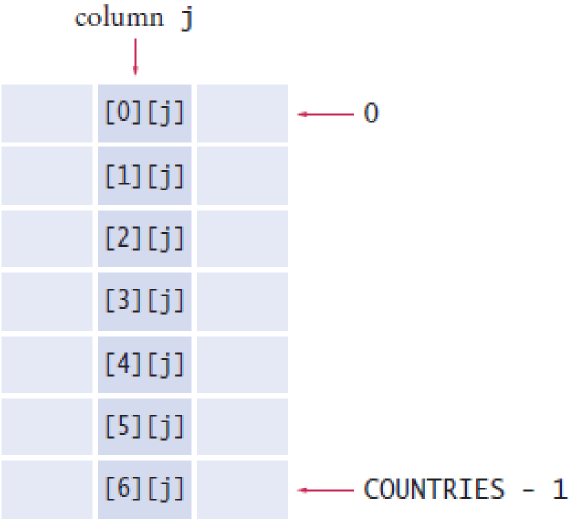
- Loop to compute the sum of the `i`th row

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

Accessing Rows and Columns

- To find the sum of the j th column
  - Form the sum of counts[i][j], where i ranges from 0 to COUNTRIES – 1

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



section_6/Medals.java

```java
/**
   This program prints a table of medal winner counts with row totals.
*/
public class Medals
{
   public static void main(String[] args)
   {
      final int COUNTRIES = 7;
      final int MEDALS = 3;

      String[] countries =
         {
            "Canada",
            "China",
            "Germany",
            "Korea",
            "Japan",
            "Russia",
            "United States"
         };

      int[][] counts =
         {
            { 1, 0, 1 },
            { 1, 1, 0 },
            { 0, 0, 1 },
            { 1, 0, 0 },
            { 0, 1, 1 },
            { 0, 1, 1 },
            { 1, 1, 0 }
         };

```

Continued

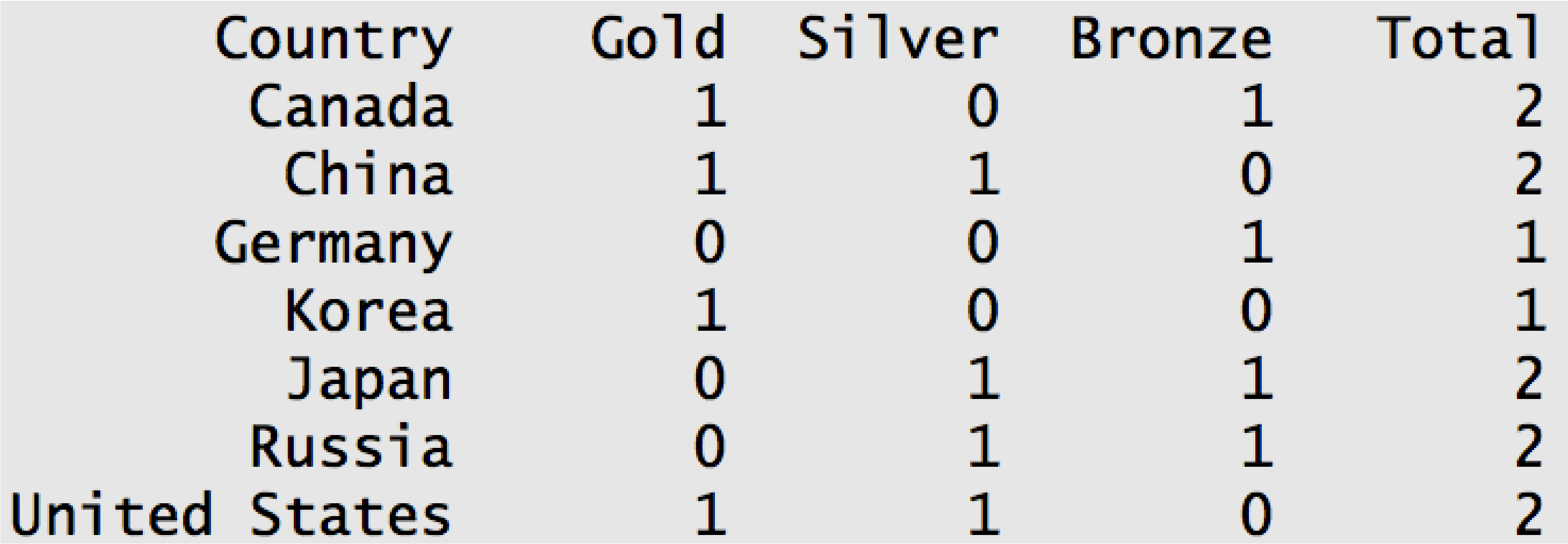
section_6/Medals.java

```
      System.out.println("        Country    Gold  Silver  Bronze   Total");

      // Print countries, counts, and row totals
      for (int i = 0; i < COUNTRIES; i++)
      {
         // Process the ith row
         System.out.printf("%15s", countries[i]);

         int total = 0;

         // Print each row element and update the row total
         for (int j = 0; j < MEDALS; j++)
         {
            System.out.printf("%8d", counts[i][j]);
            total = total + counts[i][j];
         }

         // Display the row total and print a new line
         System.out.printf("%8d\n", total);
      }
   }
}
```

Continued

section_6/Medals.java

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 6.31

Consider an 8 × 8 array for a board game:

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

Using two nested loops, initialize the board so that zeros 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.

Continued

Self Check 6.31

Answer:

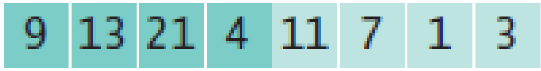
```
for (int i = 0; i < 8; i++)
{
   for (int j = 0; j < 8; j++)
   {
      board[i][j] = (i + j) % 2;
   }
}
```

Problem Solving: Discovering Algorithms by Manipulating Physical Objects

- Manipulating physical objects can give you ideas for discovering algorithms.

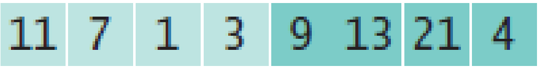
© JenCon/iStockphoto.

- The Problem: You are given an array whose size is an even number, and you are to switch the first and the second half.
- Example
  - This array



9	13	21	4	11	7	1	3

  - will become



11	7	1	3	9	13	21	4

Problem Solving: Discovering Algorithms by Manipulating Physical Objects

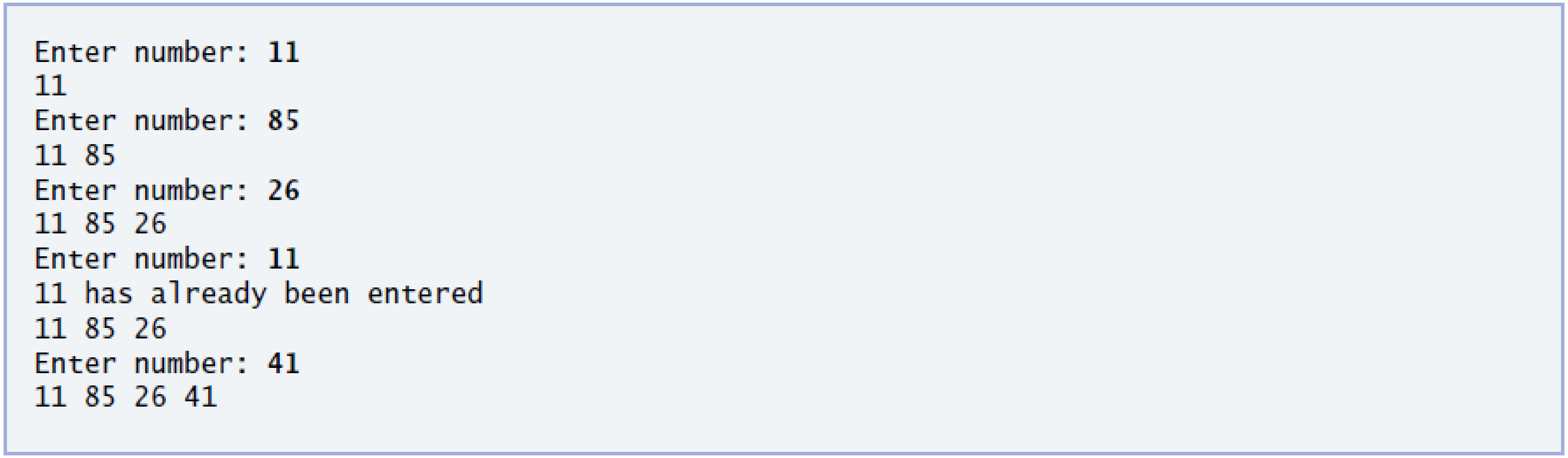
- The pseudocode

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

Duplicate Elimination

- Initialize the integer array numbers to hold five numbers between 10 and 100.
- Remember to validate the input and display an error message if the user inputs invalid data.
- If the number entered is not unique, display a message to the user; otherwise, store the number in the array and display the list of unique numbers entered so far.

Sample Output



```
Enter number: 11
11
Enter number: 85
11 85
Enter number: 26
11 85 26
Enter number: 11
11 has already been entered
11 85 26
Enter number: 41
11 85 26 41
```

Rotation

- Write a method that is passed an array, x, of doubles and an integer rotation amount, n.
- The method creates a new array with the items of x moved forward by n positions.
- Elements that are rotated off the array will appear at the end.
- For example, suppose x contains the following items in sequence:

  1 2 3 4 5 6 7
- After rotating by 3, the elements in the new array will appear in this sequence:

  5 6 7 1 2 3 4
- Array x should be left unchanged by this method.

Peevish Postman Problem

- A postman works in a small post office with consecutive letter boxes numbered 1 to 100.
- Each box was equipped with a door that could be opened and closed.
- Late one evening the postman made a “pass” through the boxes and opened every door.
- Still bored, he walked back to the beginning and made a second pass, this time visiting boxes 2, 4, 6, …, 100.
- Since those doors were now open, he closed them.
- On the third pass he visited boxes 3, 6, 9, 12, …, 99 and if a door was open he closed it, and if the door was closed he opened it.
- He continued to make passes through the boxes and always followed the same rule:
- On each pass *i* from 1 to 100, he visited only boxes that were multiples of *i*, … and changed the state of each door he visited.
- After making 100 passes at the doors, he surveyed the results and was surprised by the pattern of doors that he saw.

Peevish Postman Problem - Hint

- Use a Boolean array to represent the doors.
- A true value in the array represents an open door, and a false value represents a closed one.
- You will have to write two nested loops in order to manipulate the array as described above.
- The inner loop will control the door number visited on a single pass, and the outer loop will control the number of passes.
- Print the state of each door after the 100th pass.