```
Class C{
        int a;
}
```

what variables below are objects :

a. C c;

b. String str;

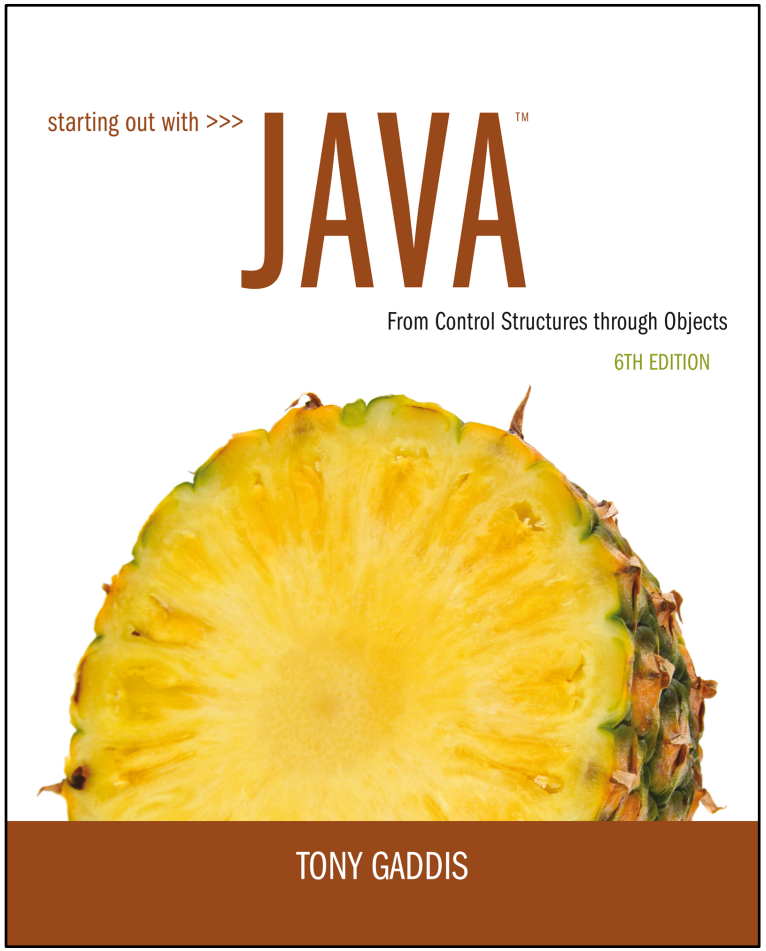
c. Scanner scanner;

d. int num;

e. float f;

Starting Out with Java: From Control Structures Through Objects

Sixth Edition



Chapter 7

Arrays

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Chapter Topics (1 of 2)

7.1 Introduction to Arrays

7.2 Processing Array Contents

7.3 Passing Arrays as Arguments to Methods

7.4 Some Useful Array Algorithms and Operations

7.5 Returning Arrays from Methods

7.6 String Arrays

7.7 Arrays of Objects

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Chapter Topics (2 of 2)

~~7.8 The Sequential Search Algorithm~~

~~7.9 Parallel Arrays~~

7.10 Two-Dimensional Arrays

7.11 Arrays with Three or More Dimensions

~~7.12 The Selection Sort and the Binary Search~~

~~7.13 Command-Line Arguments~~

~~7.14 The ArrayList Class~~

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

7.1 Introduction to Arrays

- Primitive variables are designed to hold only one value at a time.
- Arrays allow us to create a collection of like values that are indexed.
- An array can store any type of data but only one type of data at a time.
- An array is a list of data elements.

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Creating Arrays (1 of 3)

- An array is an object so it needs an object reference.

```
// Declare a reference to an array that will hold integers.
int[] numbers;
```

- The next step creates the array and assigns its address to the `numbers` variable.

```
// Create a new array that will hold 6 integers.
numbers = new int[6];
```

Array element values are initialized to 0.
Array indexes always start at 0.

Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Creating Arrays (2 of 3)

- It is possible to declare an array reference and create it in the same statement.

```
int[] numbers = new int[6];
```

- Arrays may be of any type.

```
float[] temperatures = new float[100];
char[] letters = new char[41];
long[] units = new long[50];
double[] sizes = new double[1200];
```

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Creating Arrays (3 of 3)

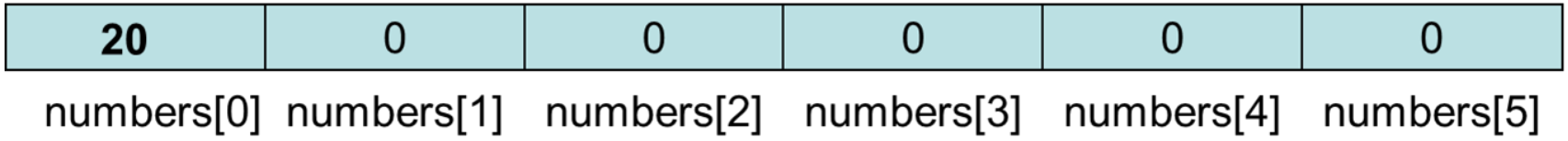
- The array size must be a non-negative number.
- It may be a literal value, a constant, or variable.

```
final int ARRAY_SIZE = 6;
int[] numbers = new int[ARRAY_SIZE];
```

- Once created, an array size is fixed and cannot be changed.

Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Accessing the Elements of an Array



- An array is accessed by:
  - the reference name
  - a subscript that identifies which element in the array to access.

```
numbers[0] = 20; //pronounced "numbers sub zero"
```

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Inputting and Outputting Array Elements

- Array elements can be treated as any other variable.
- They are simply accessed by the same name and a subscript.
- See example: ArrayDemo1.java
- Array subscripts can be accessed using variables (such as for loop counters).
- See example: ArrayDemo2.java

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Bounds Checking

- Array indexes always start at zero and continue to (array length − 1).

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

- This array would have indexes 0 through 9.
- See example: InvalidSubscript.java
- In `for` loops, it is typical to use *i*, *j,* and *k* as counting variables.
  - It might help to think of *i* as representing the word **index**.

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Off-By-One Errors

- It is very easy to be off-by-one when accessing arrays.

```
// This code has an off-by-one error.
int[] numbers = new int[100];
for (int i = 1; i <= 100; i++)
   numbers[i] = 99;
```

- Here, the equal sign allows the loop to continue on to index 100, where 99 is the last index in the array.
- This code would throw an `ArrayIndexOutOfBoundsException`.

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Array Initialization

- When relatively few items need to be initialized, an initialization list can be used to initialize the array.

```
int[]days = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

- The numbers in the list are stored in the array in order:
  - `days[0]` is assigned 31,
  - `days[1]` is assigned 28,
  - `days[2]` is assigned 31,
  - `days[3]` is assigned 30,
  - etc.
- See example: ArrayInitialization.java

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Alternate Array Declaration

- Previously we showed arrays being declared:

```
int[] numbers;
```

  - However, the brackets can also go here:

```
int numbers[];
```

  - These are equivalent but the first style is typical.
- Multiple arrays can be declared on the same line.

```
int[] numbers, codes, scores;
```

- With the alternate notation each variable must have brackets.

```
int numbers[], codes[], scores;
```

  - The `scores` variable in this instance is simply an `int` variable.

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

7.2 Processing Array Contents (1 of 2)

- Processing data in an array is the same as any other variable.

```
grossPay = hours[3] * payRate;
```

- Pre and post increment works the same:

```
int[] score = {7, 8, 9, 10, 11};
++score[2]; // Pre-increment operation
score[4]++; // Post-increment operation
```

- See example: PayArray.java

Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

7.2 Processing Array Contents (2 of 2)

- Array elements can be used in relational operations:

```
if(cost[20] < cost[0])
{
   //statements
}
```

- They can be used as loop conditions:

```
while(value[count] != 0)
{
   //statements
}
```

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Array Length

- Arrays are objects and provide a public field named `length` that is a constant that can be tested.

```
double[] temperatures = new double[25];
```

  - The length of this array is 25.

- The length of an array can be obtained via its `length` constant.

```
int size = temperatures.length;
```

  - The variable `size` will contain 25.

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

The Enhanced `for` Loop (1 of 2)

- Simplified array processing (read only)
- Always goes through all elements
- General format:

```
for(datatype elementVariable : array)
   statement;
```

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

The Enhanced `for` Loop (2 of 2)

Example:

```
int[] numbers = {3, 6, 9};
For(int val : numbers)
{
      System.out.println("The next value is " +
                         val);
}
```

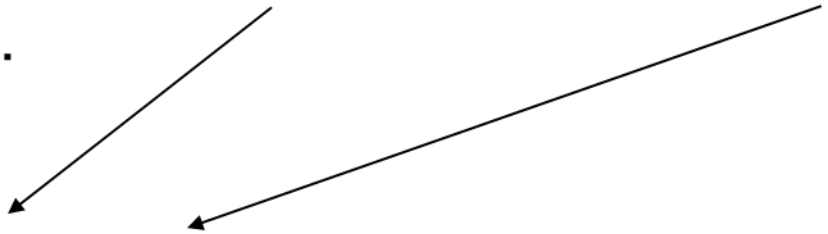
Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Array Size (1 of 2)

- The `length` constant can be used in a loop to provide automatic bounding.

Index subscripts start at 0 and end at one **less than** the array length.



```
for(int i = 0; i < temperatures.length; i++)
{
  System.out.println("Temperature " + i ": "
                              + temperatures[i]);
}
```

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Array Size (2 of 2)

- You can let the user specify the size of an array:

```
int numTests;
int[] tests;
Scanner keyboard = new Scanner(System.in);
System.out.print("How many tests do you have? ");
numTests = keyboard.nextInt();
tests = new int[numTests];
```

- See example: DisplayTestScores.java

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Reassigning Array References (1 of 3)

- An array reference can be assigned to another array of the same type.

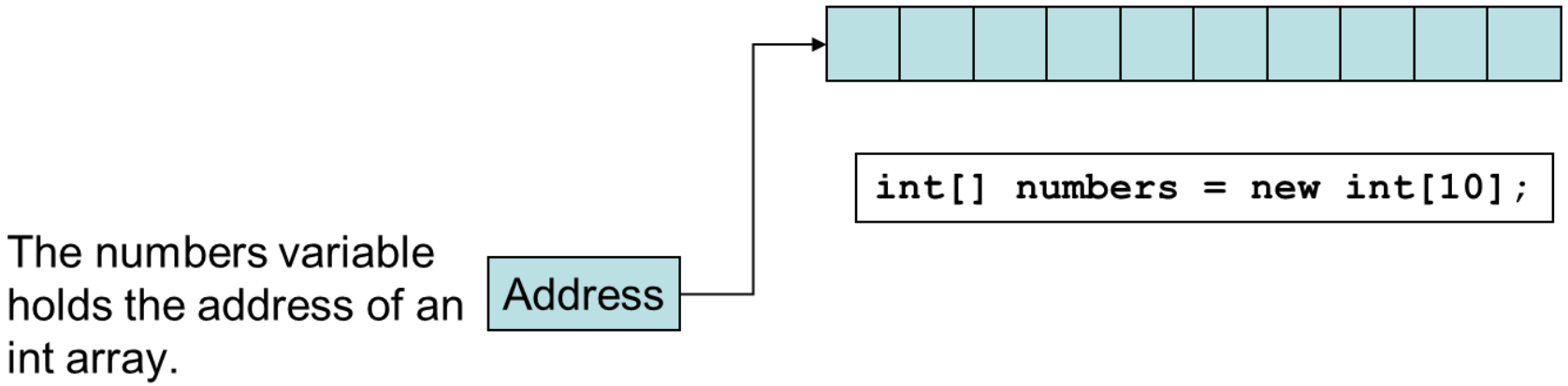
```
// Create an array referenced by the numbers variable.
int[] numbers = new int[10];
// Reassign numbers to a new array.
numbers = new int[5];
```

- If the first (10 element) array no longer has a reference to it, it will be garbage collected.

Pearson

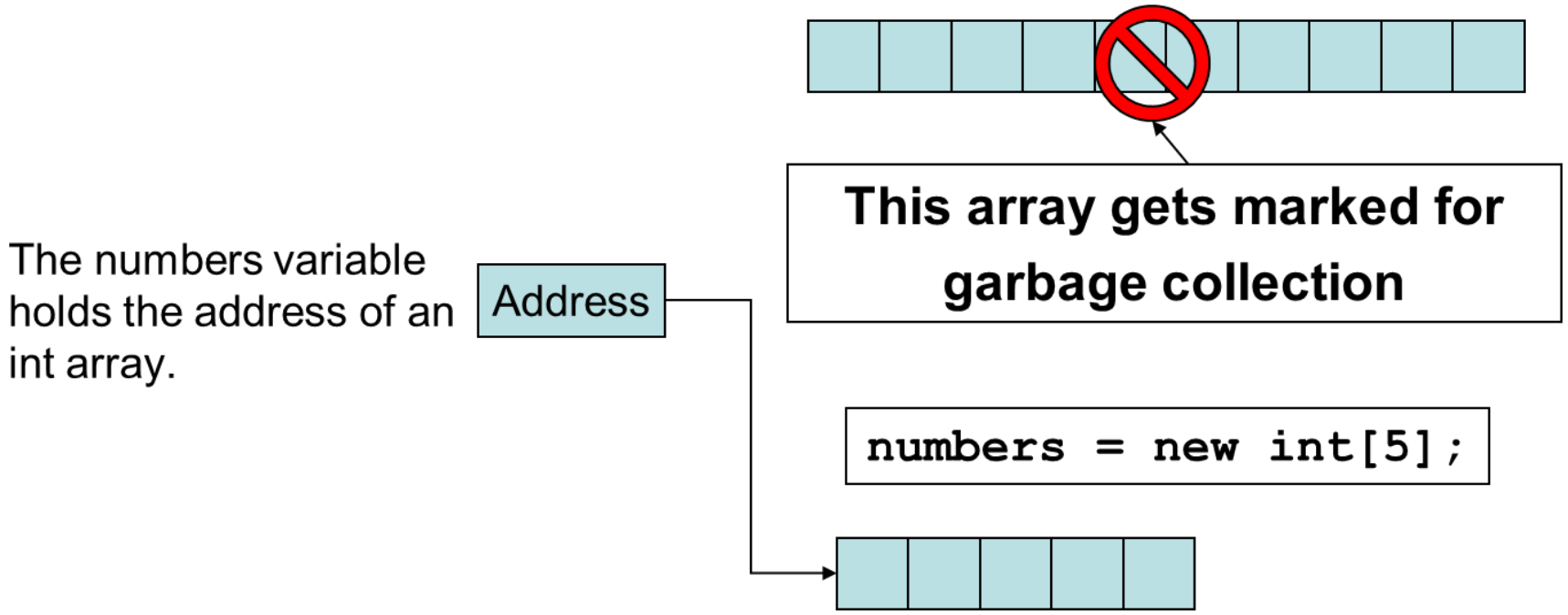
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Reassigning Array References (2 of 3)



Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Reassigning Array References (3 of 3)

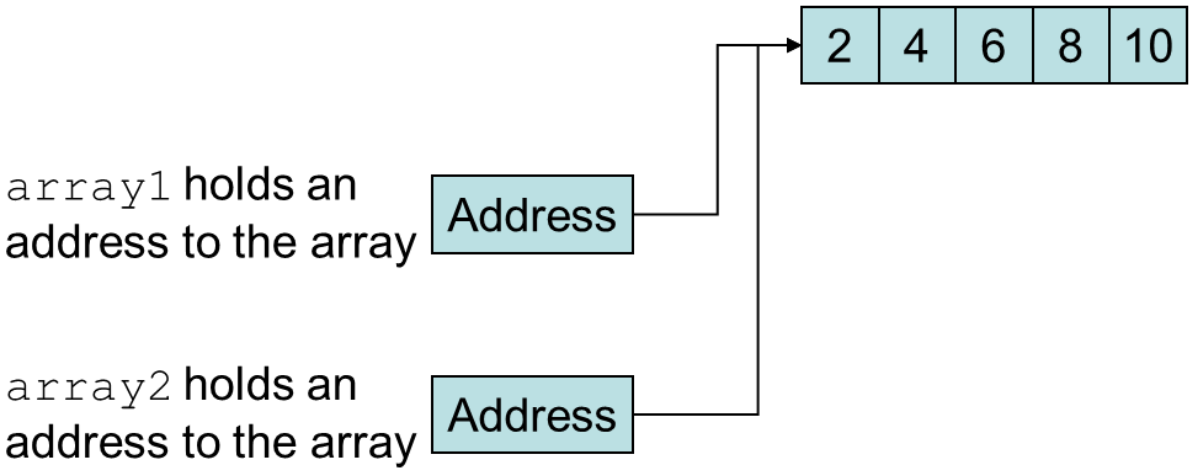


Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Copying Arrays (1 of 2)

- This is **not** the way to copy an array.

```
int[] array1 = { 2, 4, 6, 8, 10 };
int[] array2 = array1; // This does not copy array1.
```



Example: SameArray.java

Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Copying Arrays (2 of 2)

- You cannot copy an array by merely assigning one reference variable to another.
- You need to copy the individual elements of one array to another.

```
int[] firstArray = {5, 10, 15, 20, 25 };
int[] secondArray = new int[5];
for (int i = 0; i < firstArray.length; i++)
  secondArray[i] = firstArray[i];
```

- This code copies each element of `firstArray` to the corresponding element of `secondArray`.

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Comparing Arrays

- The == operator determines only whether array references point to the same array object.

```
int[] firstArray = { 5, 10, 15, 20, 25 };
int[] secondArray = { 5, 10, 15, 20, 25 };

if (firstArray == secondArray) // This is a mistake.
   System.out.println("The arrays are the same.");
else
   System.out.println("The arrays are not the same.");
```

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Comparing Arrays: Example

```
int[] firstArray = { 2, 4, 6, 8, 10 };
int[] secondArray = { 2, 4, 6, 8, 10 };
boolean arraysEqual = true;
int i = 0;

// First determine whether the arrays are the same size.
if (firstArray.length != secondArray.length)
  arraysEqual = false;

// Next determine whether the elements contain the same
data.
while (arraysEqual && i < firstArray.length)
{
  if (firstArray[i] != secondArray[i])
    arraysEqual = false;
  i++;
}

if (arraysEqual)
  System.out.println("The arrays are equal.");
else
  System.out.println("The arrays are not equal.");
```

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Useful Array Operations (1 of 2)

- Finding the Highest Value

```
int [] numbers = new int[50];
int highest = numbers[0];
for (int i = 1; i < numbers.length; i++)
{
     if (numbers[i] > highest)
          highest = numbers[i];
}
```

- Finding the Lowest Value

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

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Useful Array Operations (2 of 2)

- Summing Array Elements:

```
int total = 0; // Initialize accumulator
for (int i = 0; i < units.length; i++)
  total += units[i];
```

- Averaging Array Elements:

```
double total = 0; // Initialize accumulator
double average; // Will hold the average
for (int i = 0; i < scores.length; i++)
      total += scores[i];
average = total / scores.length;
```

- Example: SalesData.java, Sales.java

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Passing Array Elements to a Method

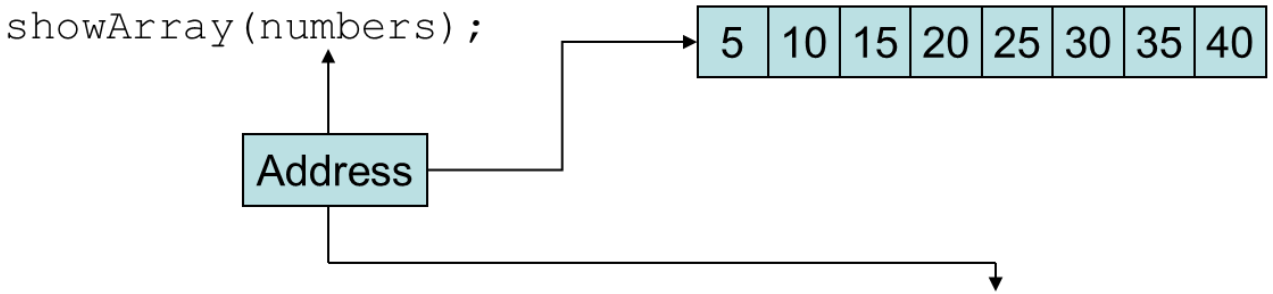
- When a single element of an array is passed to a method it is handled like any other variable.
- See example: PassElements.java
- More often you will want to write methods to process array data by passing the entire array, not just one element at a time.

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Passing Arrays as Arguments

- Arrays are objects.
- Their references can be passed to methods like any other object reference variable.




```
public static void showArray(int[] array)
{
  for (int i = 0; i < array.length; i++)
    System.out.print(array[i] + " ");
}
```

Example: PassArray.java

Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Returning an Array Reference

- A method can return a reference to an array.
- The return type of the method must be declared as an array of the right type.

```
public static double[] getArray()
{
  double[] array = { 1.2, 2.3, 4.5, 6.7, 8.9 };
  return array;
}
```

- The `getArray` method is a public static method that returns an array of doubles.
- See example: ReturnArray.java

Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

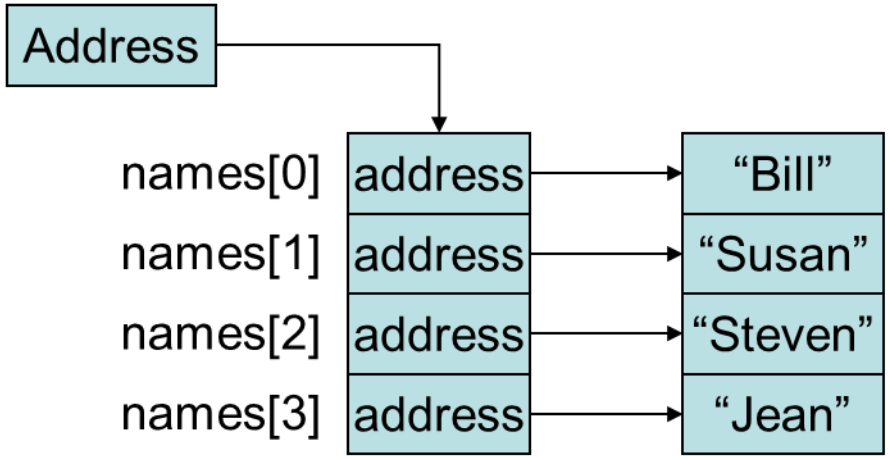
7.6 `String` Arrays (1 of 3)

- Arrays are not limited to primitive data.
- An array of String objects can be created:

```
String[] names = { "Bill", "Susan", "Steven", "Jean" };
```

The `names` variable holds the address to the array.

A `String` array is an array of references to `String` objects.



Example: MonthDays.java

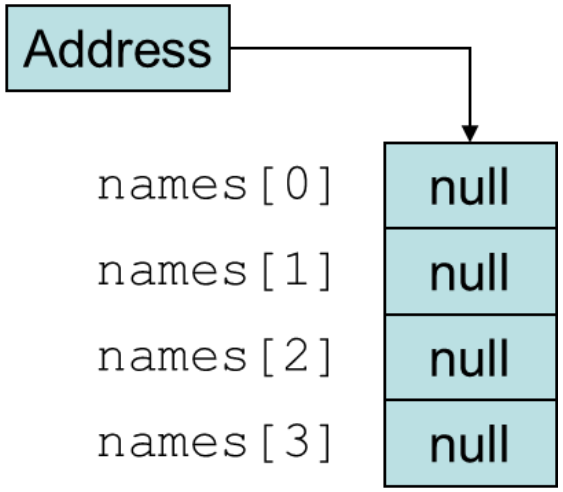
Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

7.6 `String` Arrays (2 of 3)

- If an initialization list is not provided, the `new` keyword must be used to create the array:

```
String[] names = new String[4];
```

The `names` variable holds the address to the array.



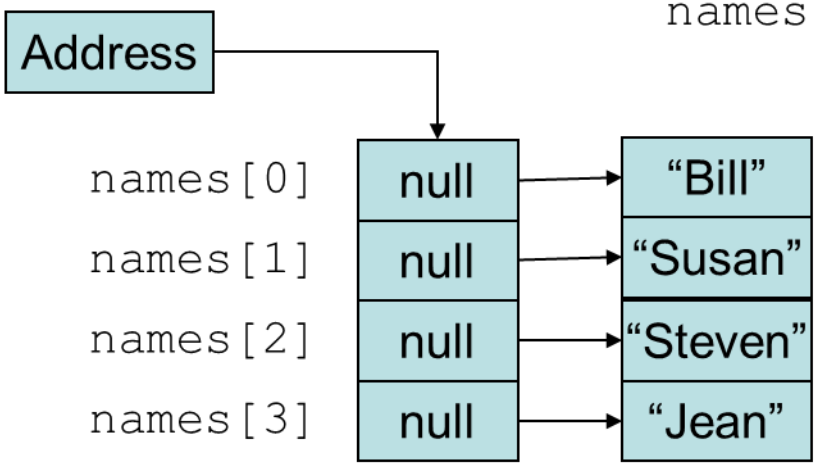
Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

7.6 `String` Arrays (3 of 3)

- When an array is created in this manner, each element of the array must be initialized.

```
names[0] = "Bill";
names[1] = "Susan";
names[2] = "Steven";
names[3] = "Jean";
```

The `names` variable holds the address to the array.



Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

The `length` Field & the `length` Method

- Arrays have a **`final field`** named **`length`.**
- String objects have a **method** named **`length`**.
- To display the length of each string held in a `String` array:

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

- An array's `length` is a **field**
  - You **do not** write a set of parentheses after its name.
- A `String`'s `length` is a **method**
  - You **do** write the parentheses after the name of the `String` class's `length` method.

Pearson

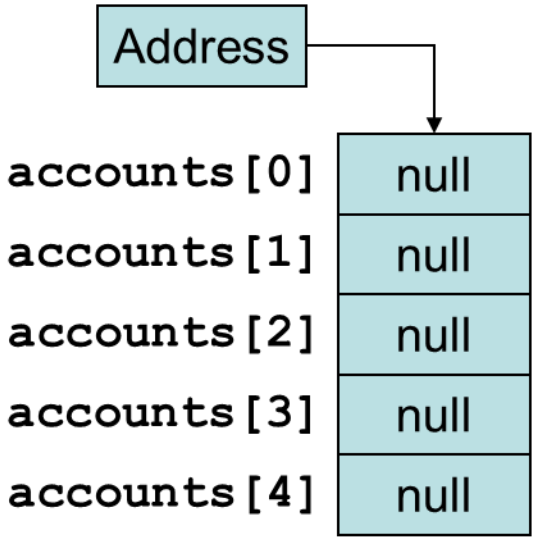
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

7.7 Arrays of Objects (1 of 2)

- Because Strings are objects, we know that arrays can contain objects.

```
BankAccount[] accounts = new BankAccount[5];
```

The `accounts` variable holds the address of an `BankAccount` array.



The array is an array of references to `BankAccount` objects.

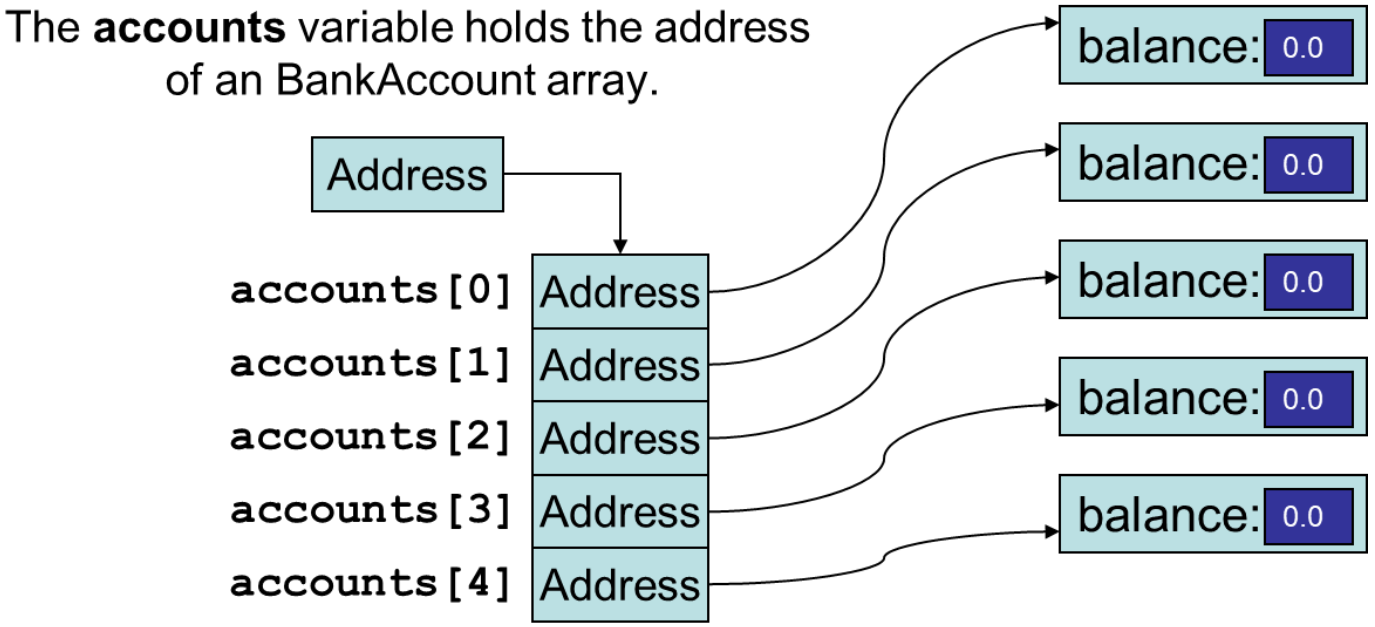
Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

7.7 Arrays of Objects (2 of 2)

- Each element needs to be initialized.

```
for (int i = 0; i < accounts.length; i++)
  accounts[i] = new BankAccount();
```

- See example: ObjectArray.java

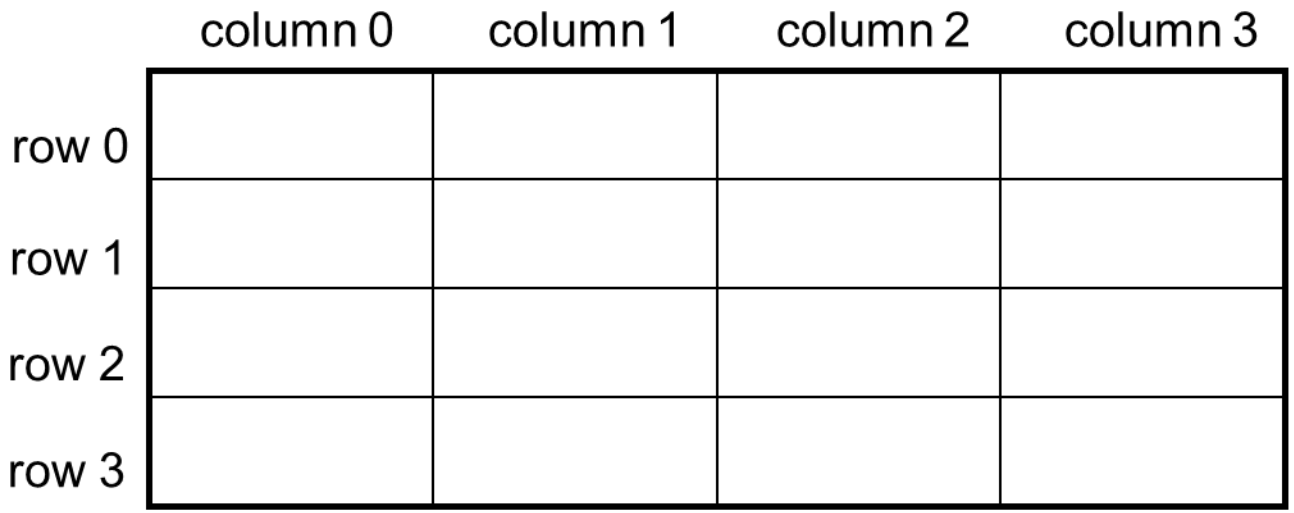


Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

7.10 Two-Dimensional Arrays (1 of 2)

- A two-dimensional array is an array of arrays.
- It can be thought of as having rows and columns.



Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

7.10 Two-Dimensional Arrays (2 of 2)

- Declaring a two-dimensional array requires two sets of brackets and two size declarators
  - The first one is for the number of rows
  - The second one is for the number of columns.

```
double[][] scores = new double[3][4];
```

- The two sets of brackets in the data type indicate that the scores variable will reference a two-dimensional array.
- Notice that each size declarator is enclosed in its own set of brackets.

Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Accessing Two-Dimensional Array Elements (1 of 5)

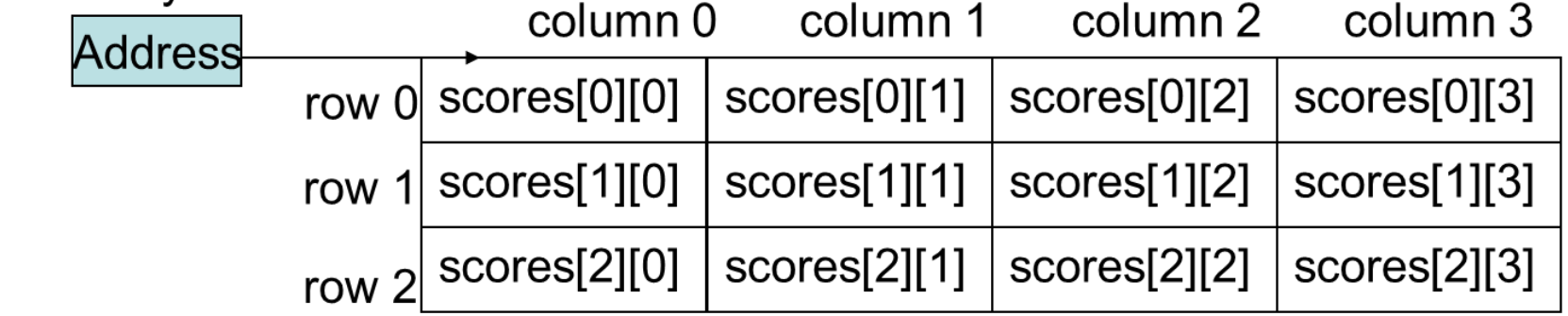
- When processing the data in a two-dimensional array, each element has two subscripts:
  - one for its row and
  - another for its column.

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Accessing Two-Dimensional Array Elements (2 of 5)

The `scores` variable holds the address of a 2D array of `doubles`.



Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Accessing Two-Dimensional Array Elements (3 of 5)

Accessing one of the elements in a two-dimensional array requires the use of both subscripts.

The `scores` variable holds the address of a 2D array of `doubles`.

```
scores[2][1] = 95;
```

Address →

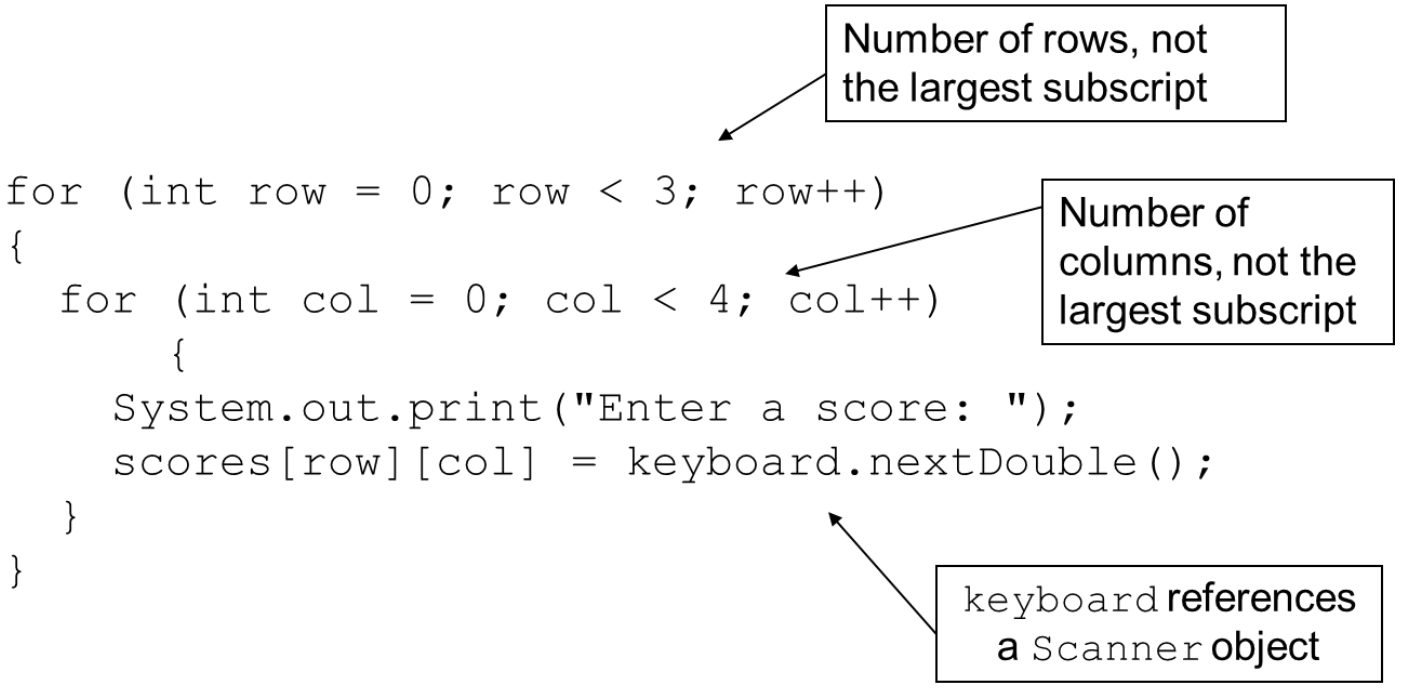
	column 0	column 1	column 2	column 3
row 0	0	0	0	0
row 1	0	0	0	0
row 2	0	**95**	0	0

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Accessing Two-Dimensional Array Elements (4 of 5)

- Programs that process two-dimensional arrays can do so with nested loops.
- To fill the scores array:



Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Accessing Two-Dimensional Array Elements (5 of 5)

- To print out the `scores` array:

```
for (int row = 0; row < 3; row++)
{
  for (int col = 0; col < 4; col++)
  {
    System.out.println(scores[row][col]);
  }
}
```

- See example: CorpSales.java

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Initializing a Two-Dimensional Array (1 of 2)

- Initializing a two-dimensional array requires enclosing each row's initialization list in its own set of braces.

```
int[][] numbers = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
```

- Java automatically creates the array and fills its elements with the initialization values.
  - row 0 {1, 2, 3}
  - row 1 {4, 5, 6}
  - row 2 {7, 8, 9}
- Declares an array with three rows and three columns.

Pearson

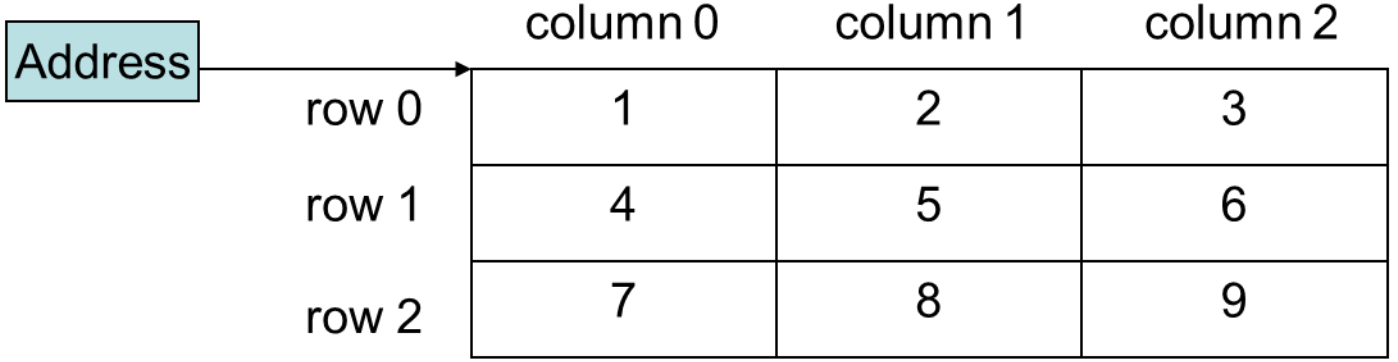
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Initializing a Two-Dimensional Array (2 of 2)

```
int[][] numbers = {{1, 2, 3},
                   {4, 5, 6},
                   {7, 8, 9}};
```

The `numbers` variable holds the address of a 2D array of `int` values.

produces:



Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

The `length` Field (1 of 2)

- Two-dimensional arrays are arrays of one-dimensional arrays.
- The length field of the array gives the number of rows in the array.
- Each row has a length constant tells how many columns is in that row.
- Each row can have a different number of columns.

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

The `length` Field (2 of 2)

- To access the length fields of the array:

```
int[][] numbers = { { 1, 2, 3, 4 },
                    { 5, 6, 7 },
                    { 9, 10, 11, 12 } };

for (int row = 0; row < numbers.length; row++)
{
  for (int col = 0; col < numbers[row].length;
col++)
    System.out.println(numbers[row][col]);
}
```

Number of rows

Number of columns in this row.

- See example: Lengths.java

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Summing the Elements of a Two-Dimensional Array

```
int[][] numbers = { { 1, 2, 3, 4 },
                    {5, 6, 7, 8},
                    {9, 10, 11, 12} };
int total;
total = 0;
for (int row = 0; row < numbers.length; row++)
{
  for (int col = 0; col < numbers[row].length; col++)
    total += numbers[row][col];
}

System.out.println("The total is " + total);
```

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Summing the Rows of a Two-Dimensional Array

```
int[][] numbers = {{ 1, 2, 3, 4},
                   {5, 6, 7, 8},
                   {9, 10, 11, 12}};
int total;

for (int row = 0; row < numbers.length; row++)
{
  total = 0;
  for (int col = 0; col < numbers[row].length; col++)
    total += numbers[row][col];
  System.out.println("Total of row "
                     + row + " is " + total);
}
```

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Summing the Columns of a Two-Dimensional Array

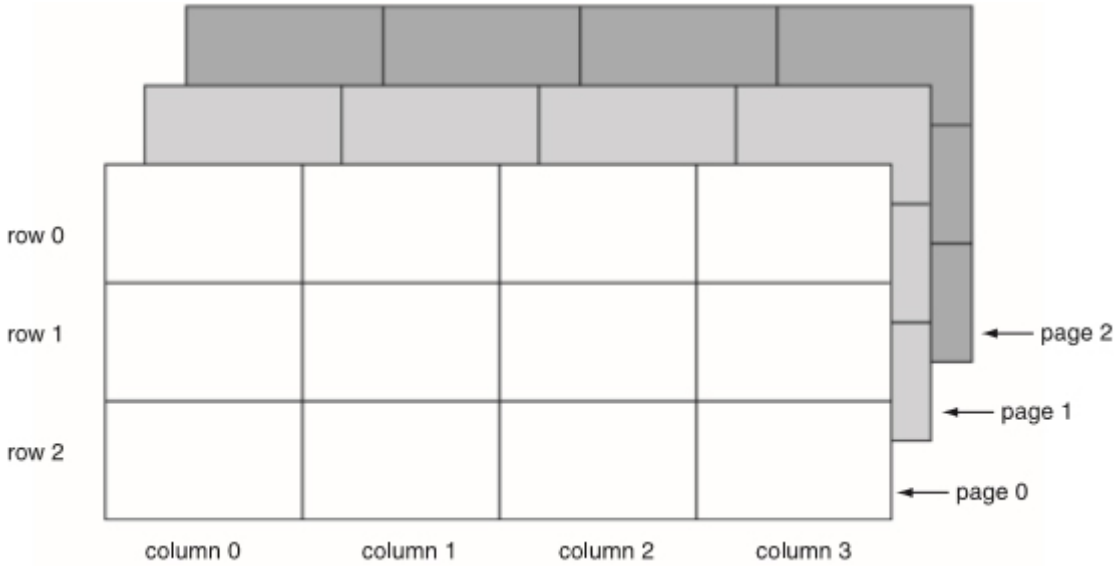
```
int[][] numbers = {{1, 2, 3, 4},
                   {5, 6, 7, 8},
                   {9, 10, 11, 12}};
int total;

for (int col = 0; col < numbers[0].length; col++)
{
  total = 0;
  for (int row = 0; row < numbers.length; row++)
    total += numbers[row][col];
  System.out.println("Total of column "
                     + col + " is " + total);
}
```

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

More Than Two Dimensions

- Java does not limit the number of dimensions that an array may be.
- More than three dimensions is hard to visualize, but can be useful in some programming problems.



Pearson
Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Copyright

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.

Pearson

Copyright © 2016, 2013, 2010 Pearson Education, Inc. All Rights Reserved