C06-Arrays and the ArrayList Class

Programming in Java

西安電子科技大學
XIDIAN UNIVERSITY

Chapter Topics (1 of 2)

WHY ARRAY?

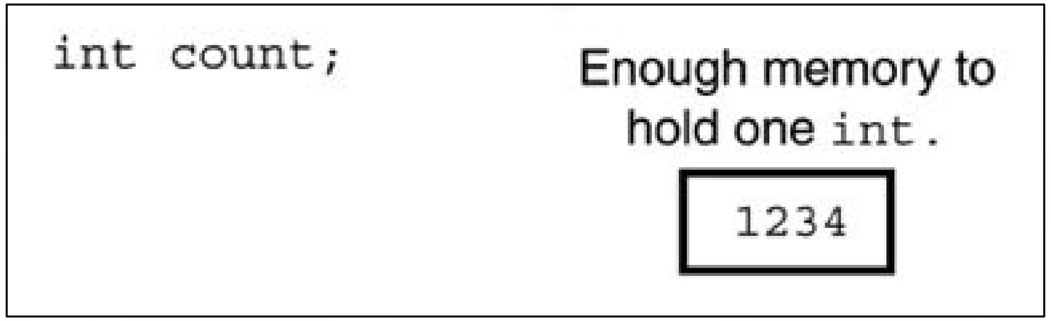
Problem?

- Implement an application that will calculate 100 students exam average.
- Variables needed?

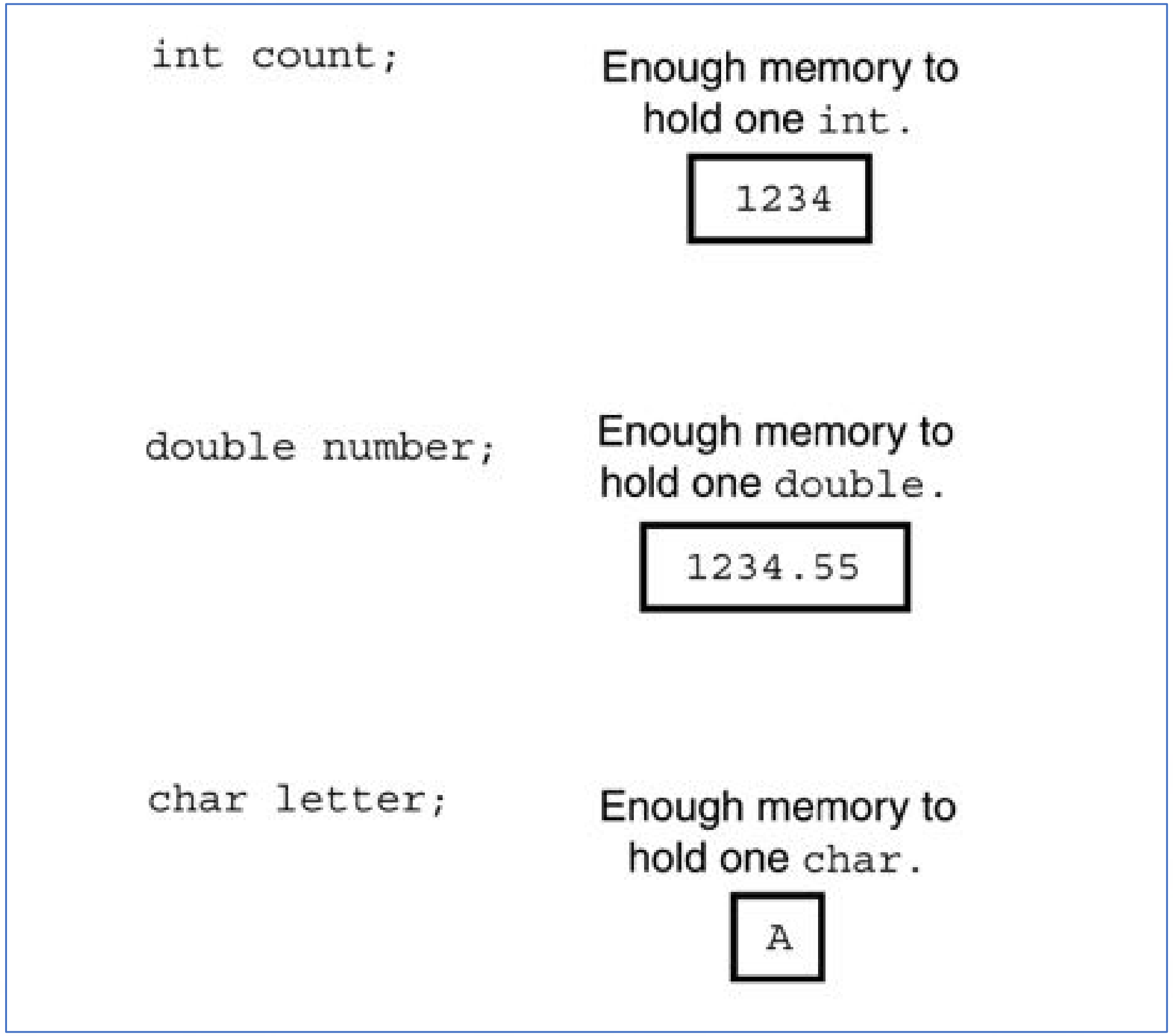
```
int student1;
int student2;
int student3;
int student4;
int student5;
.....
int student50;
int student51;
......
int student100;
```

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.


int count;
Enough memory to hold one int.
1234
double number;
Enough memory to hold one double.
1234.55
char letter;
Enough memory to hold one char.
A

Variable declarations and their memory allocations

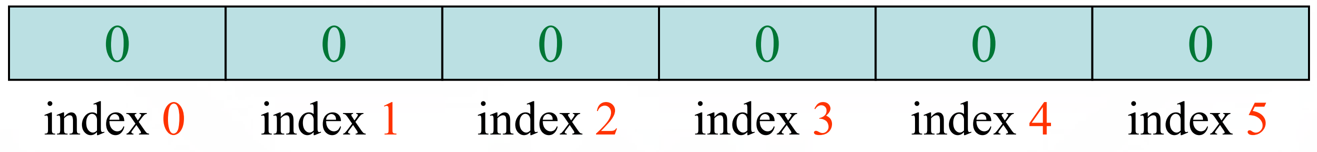
Creating Arrays (1 of 3)

- 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.

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];
```

Creating Arrays (3 of 3)

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

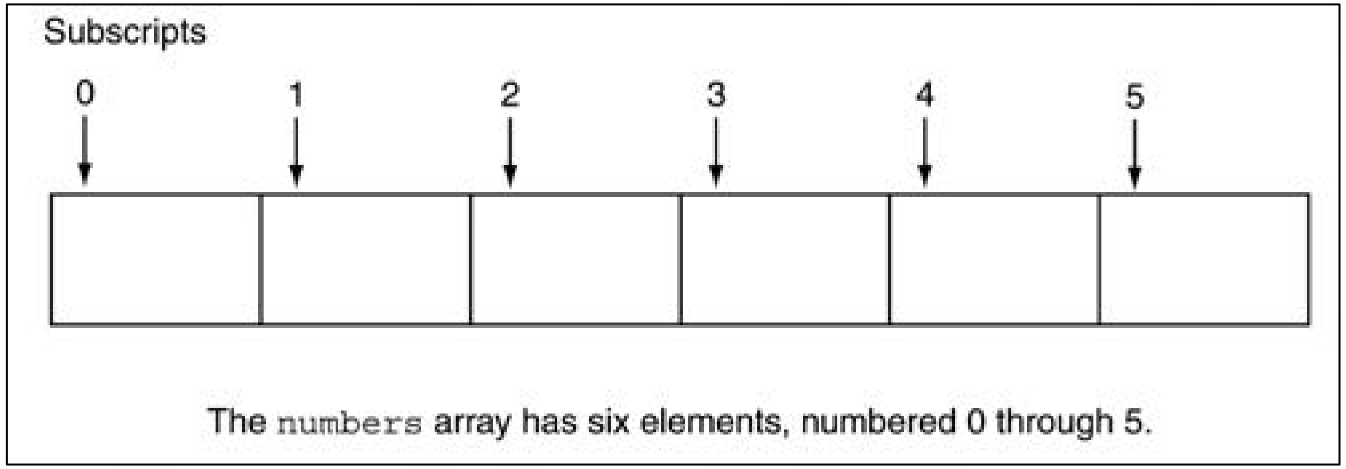
```
final int ARRAY_SIZE = 6;
int[] numbers = new int[ARRAY_SIZE];
```

$\}$ This practice can make programs easier to maintain.

- The variable should be `final` so its contents cannot be changed during the program’s execution.
- Once created, an array size is fixed and cannot be changed.

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.



The numbers array has six elements, numbered 0 through 5.

20	0	0	0	0	0
numbers[0]	numbers[1]	numbers[2]	numbers[3]	numbers[4]	numbers[5]

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

Inputting and Outputting Array Elements

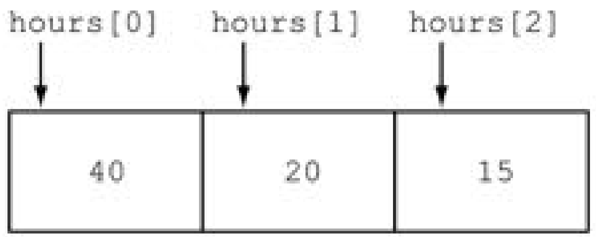
- Array elements can be treated as any other variable.
- They are simply accessed by the same name and a subscript.
- See example: ArrayDemo1.java
- Array subscripts can be accessed using variables (such as for loop counters).
- See example: ArrayDemo2.java

ArrayDemo1.java

```java
import java.util.Scanner;   // Needed for Scanner class

/**
   This program shows values being stored in an array's
   elements and displayed.
*/

public class ArrayDemo1
{
   public static void main(String[] args)
   {
      final int EMPLOYEES = 3;          // Number of employees
      int[] hours = new int[EMPLOYEES]; // Array of hours

      // Create a Scanner object for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      System.out.println("Enter the hours worked by " +

                         EMPLOYEES + " employees.");

      // Get the hours worked by employee 1.
      System.out.print("Employee 1: ");
      hours[0] = keyboard.nextInt();

      // Get the hours worked by employee 2.
      System.out.print("Employee 2: ");
      hours[1] = keyboard.nextInt();

      // Get the hours worked by employee 3.
      System.out.print("Employee 3: ");
      hours[2] = keyboard.nextInt();

      // Display the values entered.
      System.out.println("The hours you entered are:");
      System.out.println(hours[0]);
      System.out.println(hours[1]);
      System.out.println(hours[2]);
   }
}
```

Program Output with Example Input Shown in Bold

```
Enter the hours worked by 3 employees.
Employee 1: 40 [Enter]
Employee 2: 20 [Enter]
Employee 3: 15 [Enter]
The hours you entered are:
40
20
15
```

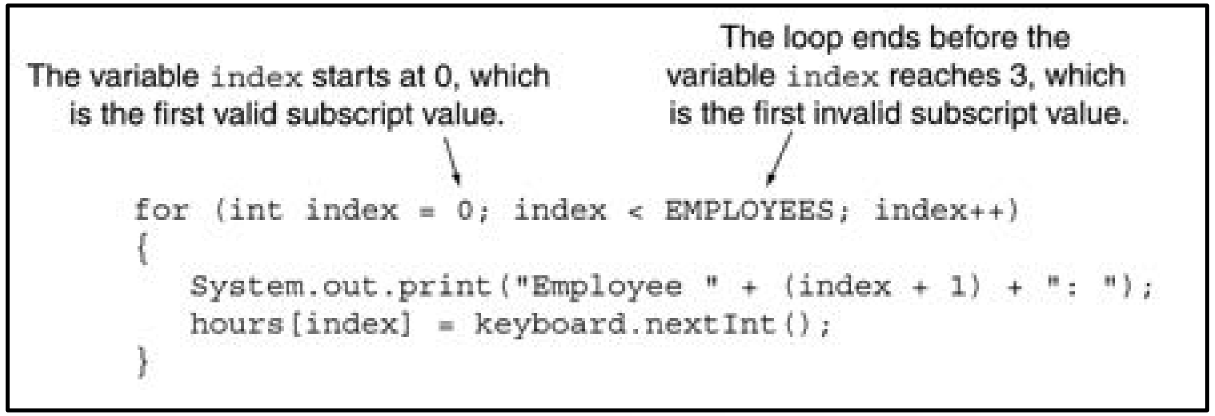


ArrayDemo2.java

```java
import java.util.Scanner;   // Needed for Scanner class

/**
   This program shows an array being processed with loops.
*/

public class ArrayDemo2
{
   public static void main(String[] args)
   {
      final int EMPLOYEES = 3;                 // Number of employees
      int[] hours = new int[EMPLOYEES];        // Array of hours

      // Create a Scanner object for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      System.out.println("Enter the hours worked by " +
                          EMPLOYEES + " employees.");

      // Get the hours for each employee.
      for (int index = 0; index < EMPLOYEES; index++)
      {
         System.out.print("Employee " + (index + 1) + ": ");
         hours[index] = keyboard.nextInt();
      }

      System.out.println("The hours you entered are:");

      // Display the values entered.
      for (int index = 0; index < EMPLOYEES; index++)
         System.out.println(hours[index]);
   }
}
```

Program Output with Example Input Shown in Bold

```
Enter the hours worked by 3 employees.
Employee 1: 40 [Enter]
Employee 2: 20 [Enter]
Employee 3: 15 [Enter]
The hours you entered are:
40
20
15
```


The variable index starts at 0, which is the first valid subscript value.
The loop ends before the variable index reaches 3, which is the first invalid subscript value.

```java
for (int index = 0; index < EMPLOYEES; index++)
{
   System.out.print("Employee " + (index + 1) + ": ");
   hours[index] = keyboard.nextInt();
}
```

Java performs 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*.

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`.

InvalidSubscript.java

```
 1  /**
 2     This program uses an invalid subscript with an array.
 3  */
 4
 5 public class InvalidSubscript
 6 {
 7    public static void main(String[] args)
 8    {
 9       int[] values = new int[3];
10
11       System.out.println("I will attempt to store four " +
12                          "numbers in a three-element array.");
13
14       for (int index = 0; index < 4; index++)
15       {
16          System.out.println("Now processing element " + index);
17          values[index] = 10;
18       }
19    }
20 }
```

Program Output

```
I will attempt to store four numbers in a three-element array.
Now processing element 0
Now processing element 1
Now processing element 2
Now processing element 3
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 3
        at InvalidSubscript.main(InvalidSubscript.java:17)
```

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};
```

- Note that you do not use the new key word when you use an initialization list. Java automatically creates the array and stores the values in the initialization list in it.
- 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

Java allows you to spread the initialization list across multiple lines.

```
double[] coins = { 0.05, 0.1, 0.25 };
double[] coins = { 0.05,
                   0.1,
                   0.25 };
```

ArrayInitialization.java

```java
/**
   This program shows an array being initialized.
*/

public class ArrayInitialization
{
   public static void main(String[] args)
   {
      int[] days = { 31, 28, 31, 30, 31, 30,
                     31, 31, 30, 31, 30, 31 };

      for (int index = 0; index < 12; index++)
      {
         System.out.println("Month " + (index + 1) +
                            " has " + days[index] +
                            " days.");
      }
   }
}
```

Program Output

```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```

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.

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

PayArray.java

```java
import java.util.Scanner; // Needed for Scanner class

/**
   This program stores in an array the hours worked by
   five employees who all make the same hourly wage.
*/

public class PayArray
{
   public static void main(String[] args)
   {
      final int EMPLOYEES = 5;   // Number of employees
      double payRate;            // Hourly pay rate
      double grossPay;           // Gross pay

      // Create an array to hold employee hours.
      int[] hours = new int[EMPLOYEES];

      // Create a Scanner object for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      // Get the hours worked by each employee.
      System.out.println("Enter the hours worked by " +
                         EMPLOYEES + " employees who all earn " +
                         "the same hourly rate.");

      for (int index = 0; index < EMPLOYEES; index++)
      {
         System.out.print( "Employee #" + (index + 1) + ": ");
         hours[index] = keyboard.nextInt();
      }

      // Get the hourly pay rate.
      System.out.print("Enter the hourly rate for each employee: ");
      payRate = keyboard.nextDouble();

      // Display each employee's gross pay.
      System.out.println( "Here is each employee's gross pay:");
      for (int index = 0; index < EMPLOYEES; index++)
      {
         grossPay = hours[index] * payRate;
         System.out.println("Employee #" + (index + 1) +
                            ": $" + grossPay);
      }
   }
}
```

Program Output with Example Input Shown in Bold

```
Enter the hours worked by 5 employees who all earn the same hourly rate.
Employee #1: 10 [Enter]
Employee #2: 20 [Enter]
Employee #3: 30 [Enter]
Employee #4: 40 [Enter]
Employee #5: 50 [Enter]
Enter the hourly rate for each employee: 10 [Enter]
Here is each employee's gross pay:
Employee #1: $100.0
Employee #2: $200.0
Employee #3: $300.0
Employee #4: $400.0
Employee #5: $500.0
```

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
}
```

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.

The Enhanced `for` Loop (1 of 2)

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

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

  - *dataType elementVariable* is a variable declaration.
  - *array* is the name of an array on which you wish the loop to operate.
  - *statement* is a statement that executes during a loop iteration.

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);
}
```

You cannot use the enhanced `for` loop as follows:

- if you need to change the contents of an array element
- if you need to work through the array elements in reverse order
- if you need to access some of the array elements, but not all of them
- if you need to simultaneously work with two or more arrays within the loop
- if you need to refer to the subscript number of a particular element

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]);
}
```

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

DisplayTestScores.java

```java
import java.util.Scanner; // Needed for Scanner class

/**
   This program demonstrates how the user may specify an
   array's size.
*/

public class DisplayTestScores
{
   public static void main(String[] args)
   {
      int numTests;     // The number of tests
      int[] tests;      // Array of test scores

      // Create a Scanner object for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      // Get the number of test scores.
      System.out.print("How many tests do you have? ");
      numTests = keyboard.nextInt();

      // Create an array to hold that number of scores.
      tests = new int[numTests];

      // Get the individual test scores.
      for (int index = 0; index < tests.length; index++)
      {
         System.out.print("Enter test score " +
                          (index + 1) + ": ");
         tests[index] = keyboard.nextInt();
      }

      // Display the test scores.
      System.out.println();
      System.out.println("Here are the scores you entered:");
      for (int index = 0; index < tests.length; index++)
         System.out.print(tests[index] + " ");
   }
}
```

Program Output with Example Input Shown in Bold

```
How many tests do you have? 5 [Enter]
Enter test score 1: 72 [Enter]
Enter test score 2: 85 [Enter]
Enter test score 3: 81 [Enter]
Enter test score 4: 94 [Enter]
Enter test score 5: 99 [Enter]

Here are the scores you entered:
72 85 81 94 99
```

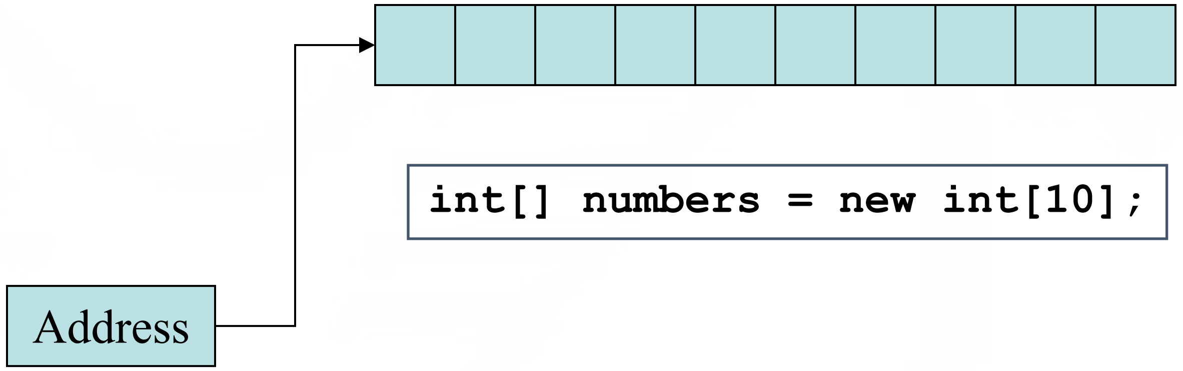
Reassigning Array References (1 of 3)

- 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.

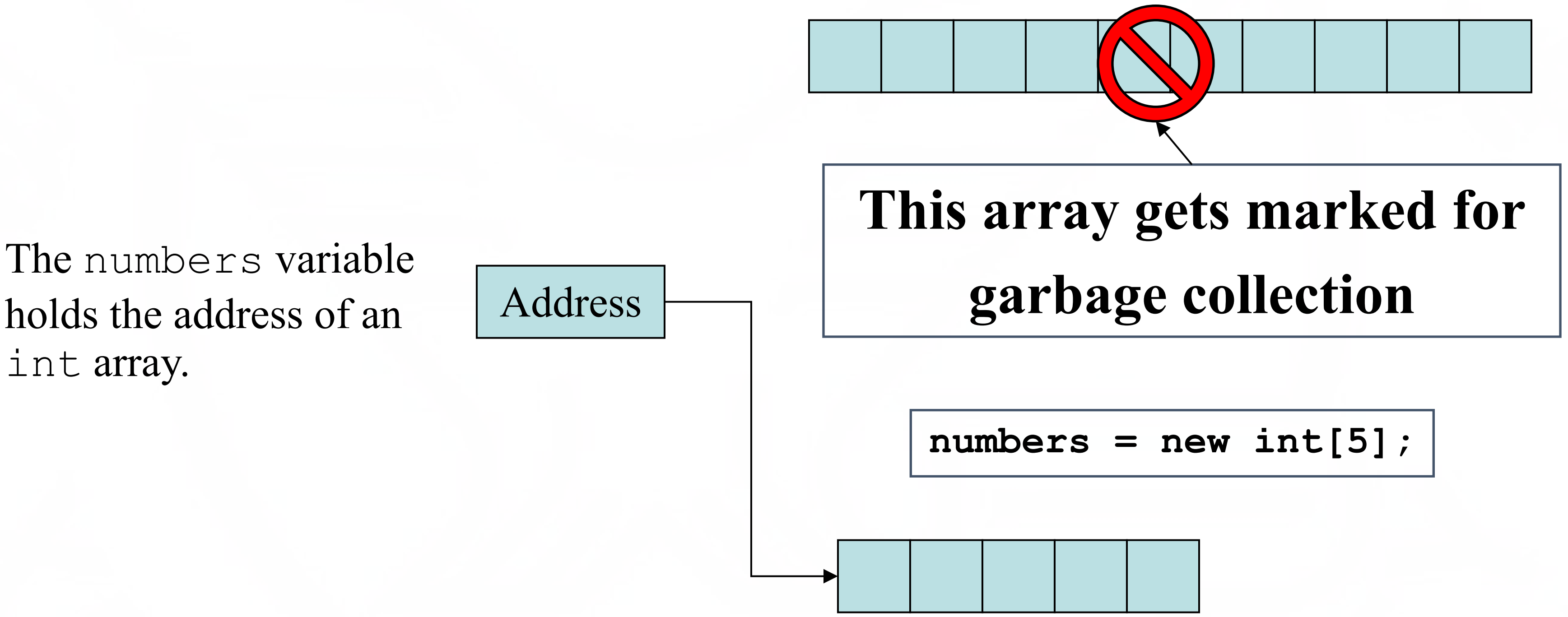
Reassigning Array References (2 of 3)



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

The numbers variable holds the address of an int array.

Reassigning Array References (3 of 3)

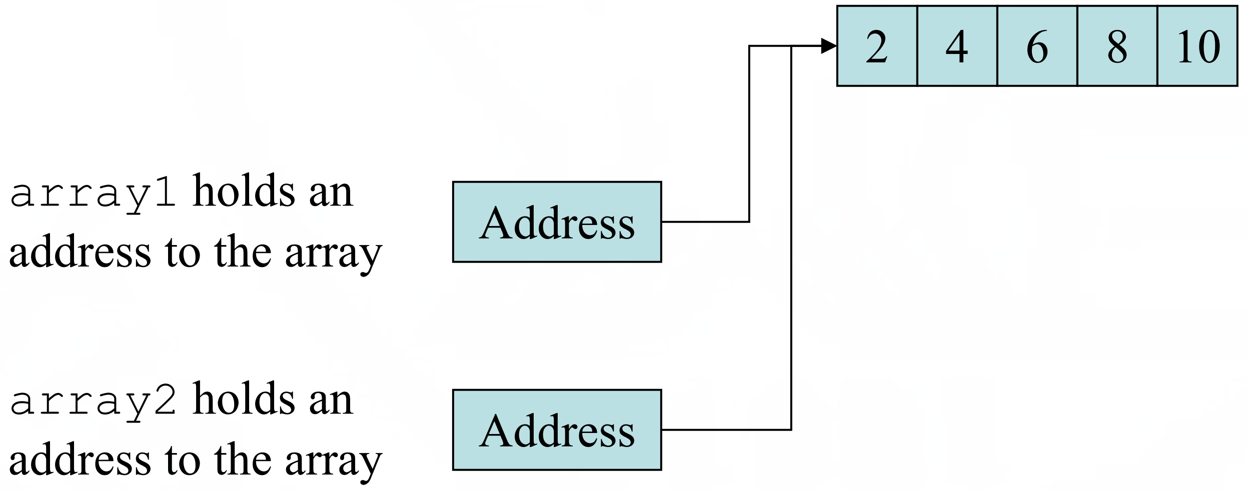

The `numbers` variable holds the address of an `int` array.
Address
This array gets marked for garbage collection
numbers = new int[5];

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.
```

- Both the array1 and array2 variables will reference the same array. This type of assignment operation is called a *reference copy*.



2
4
6
8
10
array1 holds an address to the array
Address
array2 holds an address to the array
Address

Example:
SameArray.java

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`.

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.

PassElements.java

```java
/**
   This program demonstrates passing individual array
   elements as arguments to a method.
*/

public class PassElements
{
   public static void main(String[] args)
   {
      int[] numbers = {5, 10, 15, 20, 25, 30, 35, 40};

      for (int index = 0; index < numbers.length; index++)
         showValue(numbers[index]);
   }

   /**
      The showValue method displays its argument.
      @param n The value to display.
   */

   public static void showValue(int n)
   {
      System.out.print(n + " ");
   }
}
```

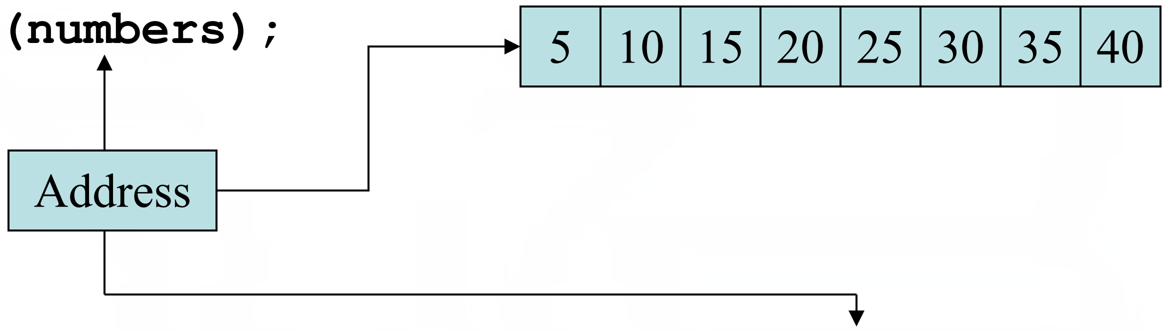
Program Output

```
5 10 15 20 25 30 35 40
```

Passing Arrays as Arguments

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

```
showArray(numbers);
```



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

Example: PassArray.java

PassArray.java

```java
import java.util.Scanner;  // Needed for Scanner class

/**
   This program demonstrates passing an array
   as an argument to a method.
*/

public class PassArray
{
   public static void main(String[] args)
   {
      final int ARRAY_SIZE = 4;  // Size of the array

      // Create an array.
      int[] numbers = new int[ARRAY_SIZE];

      // Pass the array to the getValues method.
      getValues(numbers);

      System.out.println("Here are the " +
                     "numbers that you entered:");

      // Pass the array to the showArray method.
      showArray(numbers);
   }

   /**
      The getValues method accepts a reference
      to an array as its argument. The user is
      asked to enter a value for each element.
```

```java
   */

   private static void getValues(int[] array)
   {
      // Create a Scanner objects for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      System.out.println("Enter a series of " +
                     array.length + " numbers.");

      // Read values into the array
      for (int index = 0; index < array.length; index++)
      {
         System.out.print("Enter number " +
                          (index + 1) + ": ");
         array[index] = keyboard.nextInt();
      }
   }

   /**
      The showArray method accepts an array as
      an argument and displays its contents.
      @param array A reference to the array.
   */

   public static void showArray(int[] array)
   {
      // Display the array elements.
      for (int index = 0; index < array.length; index++)
         System.out.print(array[index] + " ");
```

```java
   }
}
```

Program Output with Example Input Shown in Bold

```
Enter a series of 4 numbers.
Enter number 1: 2 [Enter]
Enter number 2: 4 [Enter]
Enter number 3: 6 [Enter]
Enter number 4: 8 [Enter]
Here are the numbers that you entered:
2 4 6 8
```

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.");
```

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.");
```

Useful Array Operations (1 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;
```

Useful Array Operations (2 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];
}
```

- Example: SalesData.java, Sales.java

SalesData.java (1/2)

```java
/**
   This class keeps the sales figures for a number of
   days in an array and provides methods for getting
   the total and average sales, and the highest and
   lowest amounts of sales.
*/

public class SalesData
{
   private double[] sales; // The sales data

   /**
      The constructor copies the elements in
      an array to the sales array.
      @param s The array to copy.
   */

   public SalesData(double[] s)
   {
      // Create an array as large as s.
      sales = new double[s.length];

      // Copy the elements from s to sales.
      for (int index = 0; index < s.length; index++)
          sales[index] = s[index];
   }

   /**
      getTotal method
      @return The total of the elements in
      the sales array.
   */

   public double getTotal()
   {
      double total = 0.0;      // Accumulator

      // Accumulate the sum of the elements
      // in the sales array.
      for (int index = 0; index < sales.length; index++)
         total += sales[index];

      // Return the total.
      return total;
   }

   /**
      getAverage method
      @return The average of the elements
      in the sales array.
   */

```

SalesData.java (2/2)

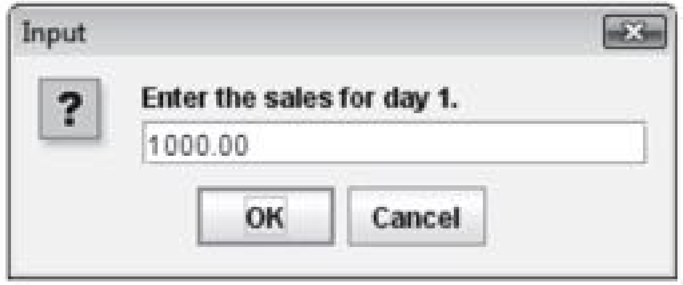
```java
   public double getAverage()
   {
      return getTotal() / sales.length;
   }

   /**
      getHighest method
      @return The highest value stored
      in the sales array.
   */

   public double getHighest()
   {
      double highest = sales[0];

      for (int index = 1; index < sales.length; index++)
      {
         if (sales[index] > highest)
            highest = sales[index];
      }

      return highest;
   }

   /**
      getLowest method
      @return The lowest value stored
      in the sales array.
   */

   public double getLowest()
   {
      double lowest = sales[0];

      for (int index = 1; index < sales.length; index++)
      {
         if (sales[index] < lowest)
            lowest = sales[index];
      }

      return lowest;
   }
}
```

(Sales.java) (1/2)

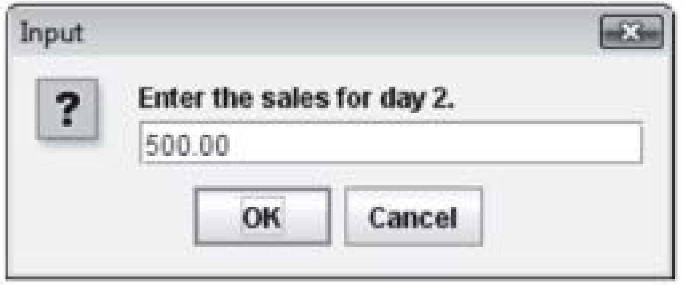
```java
import javax.swing.JOptionPane;

/**
   This program gathers sales amounts for the week.
   It uses the SalesData class to display the total,
   average, highest, and lowest sales amounts.
*/

public class Sales
{
   public static void main(String[] args)
   {
      final int ONE_WEEK = 7; // Number of elements

      // Create an array to hold sales amounts for a week.
      double[] sales = new double[ONE_WEEK];

      // Get the week's sales figures.
      getValues(sales);

      // Create a SalesData object, initialized
      // with the week's sales figures.
      SalesData week = new SalesData(sales);

      // Display the total, average, highest, and lowest
      // sales amounts for the week.
      JOptionPane.showMessageDialog(null,
         String.format("The total sales were $%,.2f\n" +
                       "The average sales were $%,.2f\n" +
                       "The highest sales were $%,.2f\n" +
                       "The lowest sales were $%,.2f",
                       week.getTotal(),
                       week.getAverage(),
                       week.getHighest(),
                       week.getLowest()));

      System.exit(0);
   }

```

(Sales.java)(1/2)

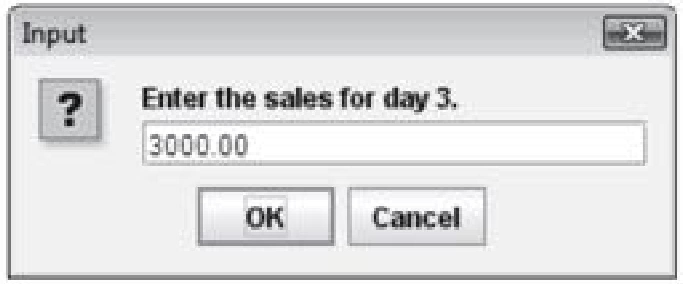
```
   /**
      The getValues method asks the user to enter sales
      amounts for each element of an array.
      @param array The array to store the values in.
   */

   private static void getValues(double[] array)
   {
      String input;  // To hold user input.

      // Get sales for each day of the week.
      for (int i = 0; i < array.length; i++)
      {
         input = JOptionPane.showInputDialog(
                     "Enter the sales for day " + (i + 1) + ".");
         array[i] = Double.parseDouble(input);
      }
   }
}
```

1


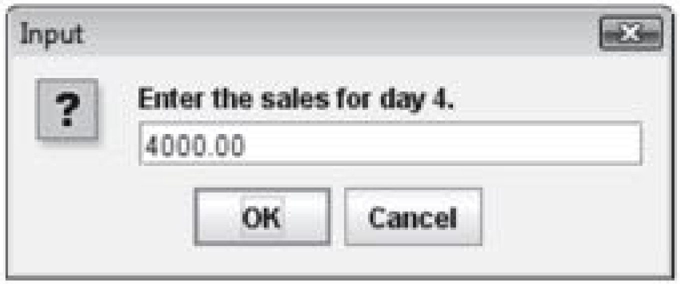
```
Enter the sales for day 1.
1000.00
```

2


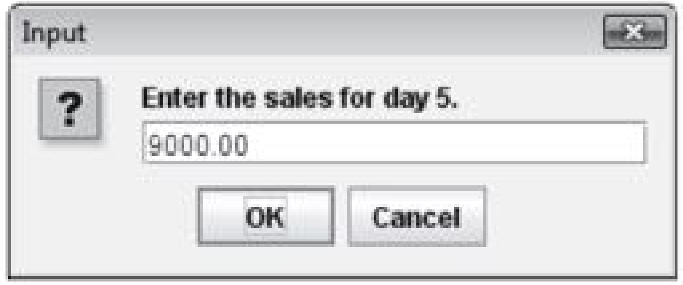
```
Enter the sales for day 2.
500.00
```

3


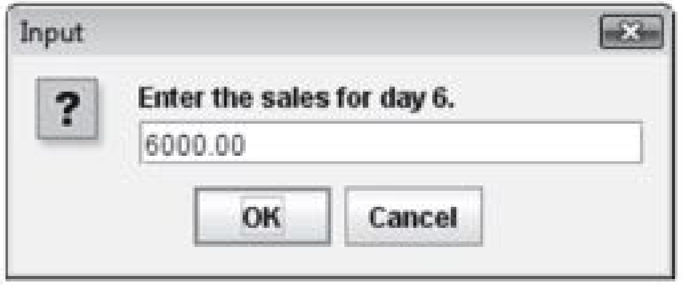
```
Enter the sales for day 3.
3000.00
```

4


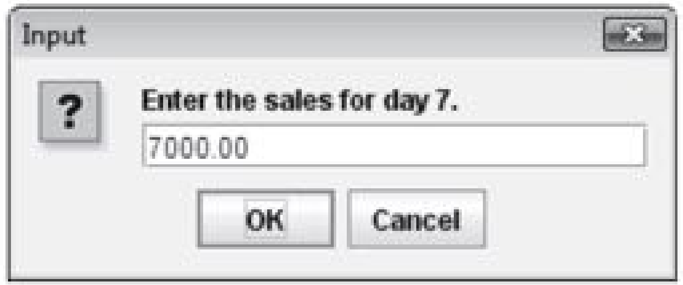
```
Enter the sales for day 4.
4000.00
```

5


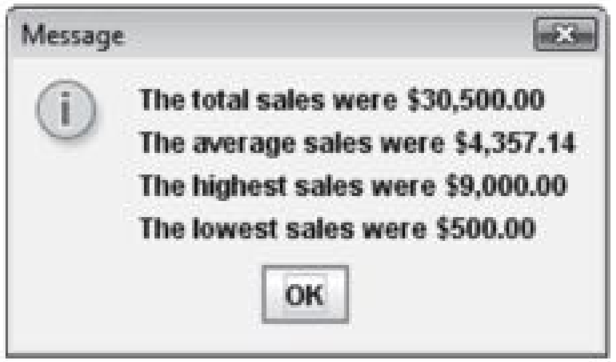
```
Enter the sales for day 5.
9000.00
```

6


```
Enter the sales for day 6.
6000.00
```

7


```
Enter the sales for day 7.
7000.00
```

8


```
The total sales were $30,500.00
The average sales were $4,357.14
The highest sales were $9,000.00
The lowest sales were $500.00
```

Partially Filled Arrays

- Typically, if the amount of data that an array must hold is unknown:
  - size the array to the largest expected number of elements.
  - use a counting variable to keep track of how much valid data is in the array.

```
…
int[] array = new int[100];
int count = 0;
…
  System.out.print("Enter a number or -1 to quit: ");
  number = keyboard.nextInt();
  while (number != -1 && count <= 99)
  {
    array[count] = number;
    count++;
    System.out.print("Enter a number or -1 to quit: ");
    number = keyboard.nextInt();
  }
…
```

`input`, `number` and `keyboard` were previously declared and `keyboard` references a `Scanner` object

Arrays and Files (1 of 2)

- Saving the contents of an array to a file:

```
int[] numbers = {10, 20, 30, 40, 50};

PrintWriter outputFile =
     new PrintWriter ("Values.txt");

for (int i = 0; i < numbers.length; i++)
  outputFile.println(numbers[i]);

outputFile.close();
```

Arrays and Files (2 of 2)

- Reading the contents of a file into an array:

```
  final int SIZE = 5; // Assuming we know the size.
int[] numbers = new int[SIZE];
int i = 0;
File file = new File ("Values.txt");
Scanner inputFile = new Scanner(file);
while (inputFile.hasNext() && i < numbers.length)
{
  numbers[i] = inputFile.nextInt();
  i++;
}
inputFile.close();
```

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

ReturnArray.java

```java
/**
   This program demonstrates how a reference to an
   array can be returned from a method.
*/

public class ReturnArray
{
   public static void main(String[] args)
   {
      double[] values;

      values = getArray();
      for (double num : values)
         System.out.print(num + " ");
   }

   /**
      getArray method
      @return A reference to an array of doubles.
   */

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

      return array;
   }
}
```

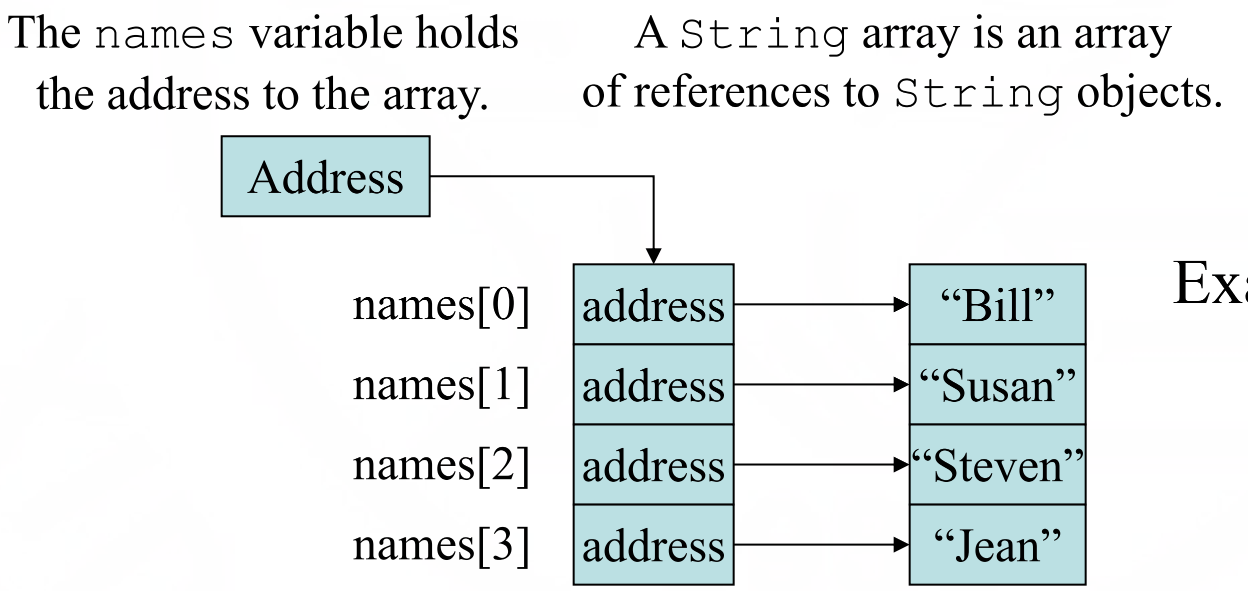
Program Output

```
1.2  2.3  4.5  6.7  8.9
```

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.
Address
names[0]
names[1]
names[2]
names[3]
address
address
address
address
“Bill”
“Susan”
“Steven”
“Jean”

Example:

[MonthDays.java]

MonthDays.java

```java
/**
   This program demonstrates an array of String objects.
*/

public class MonthDays
{
   public static void main(String[] args)
   {
      String[] months = { "January", "February", "March",
                          "April", "May", "June", "July",
                          "August", "September", "October",
                          "November", "December" };

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

      for (int index = 0; index < months.length; index++)
      {
         System.out.println(months[index] + " has " +
                            days[index] + " days.");
      }
   }
}
```

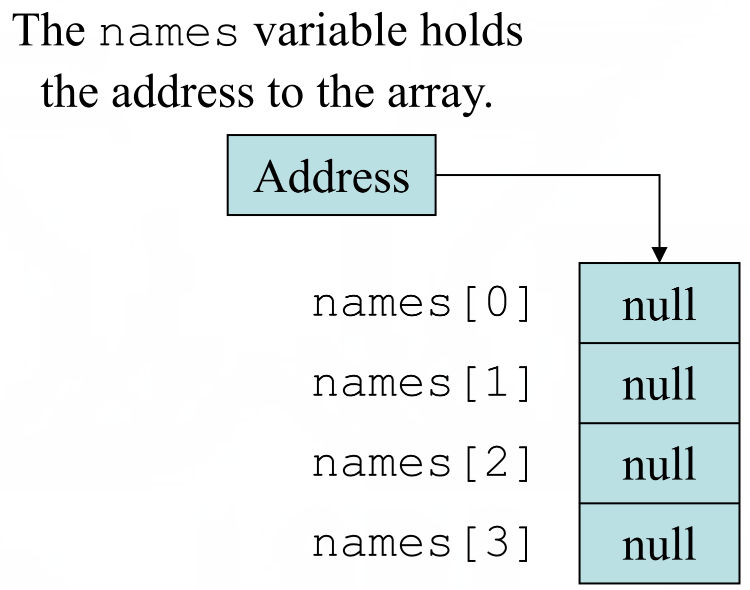
Program Output

```
January has 31 days.
February has 28 days.
March has 31 days.
April has 30 days.
May has 31 days.
June has 30 days.
July has 31 days.
August has 31 days.
September has 30 days.
October has 31 days.
November has 30 days.
December has 31 days.
```

`String` Arrays (2 of 3)

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

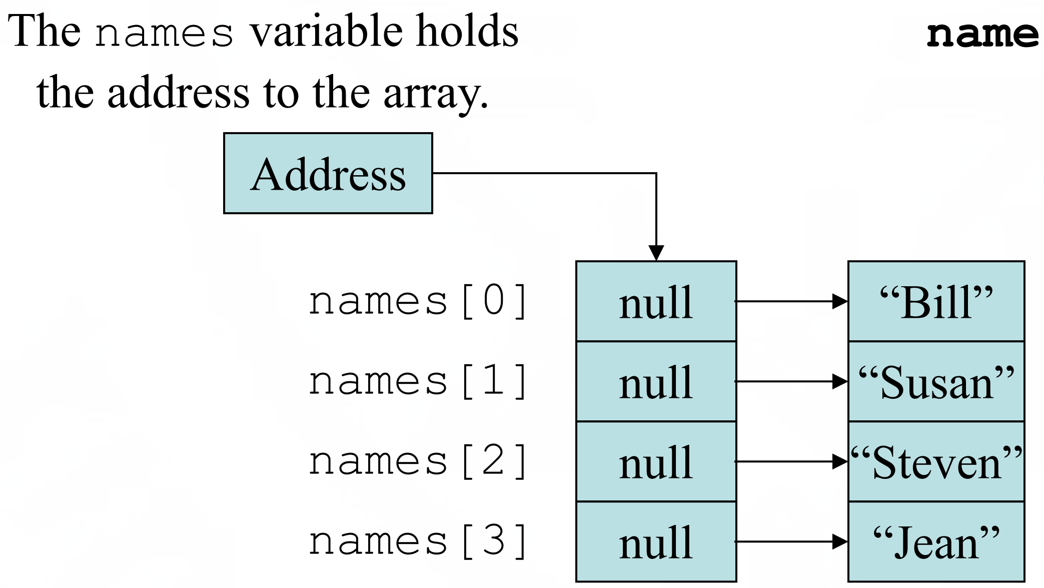
```
String[] names = new String[4];
```



String Arrays (3 of 3)

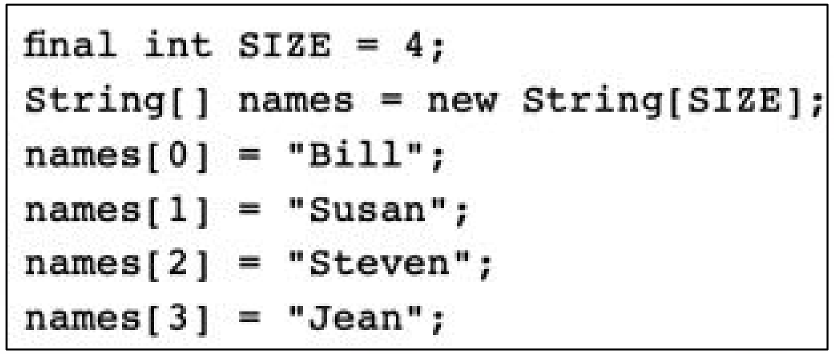
- When an array is created in this manner, each element of the array must be initialized.

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



Calling `String` Methods On Array Elements

- `String` objects have several methods, including:
  - `toUpperCase`
  - `compareTo`
  - `equals`
  - `charAt`



```
final int SIZE = 4;
String[] names = new String[SIZE];
names[0] = "Bill";
names[1] = "Susan";
names[2] = "Steven";
names[3] = "Jean";
```

- Each element of a `String` array is a `String` object.
- Methods can be used by using the array name and index as before.

```
System.out.println(names[0].toUpperCase());
char letter = names[3].charAt(0);
```

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:

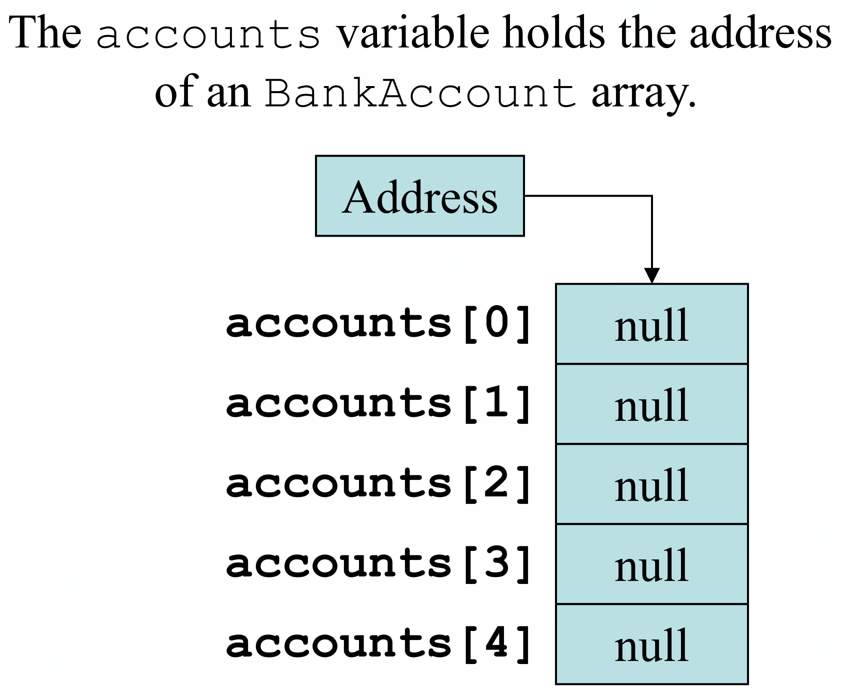
```
for (int i = 0; i < names.length; i++)
   System.out.println(names[i].length());
```

- 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.

Arrays of Objects (1 of 2)

- Because `String`s are objects, we know that arrays can contain objects.

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



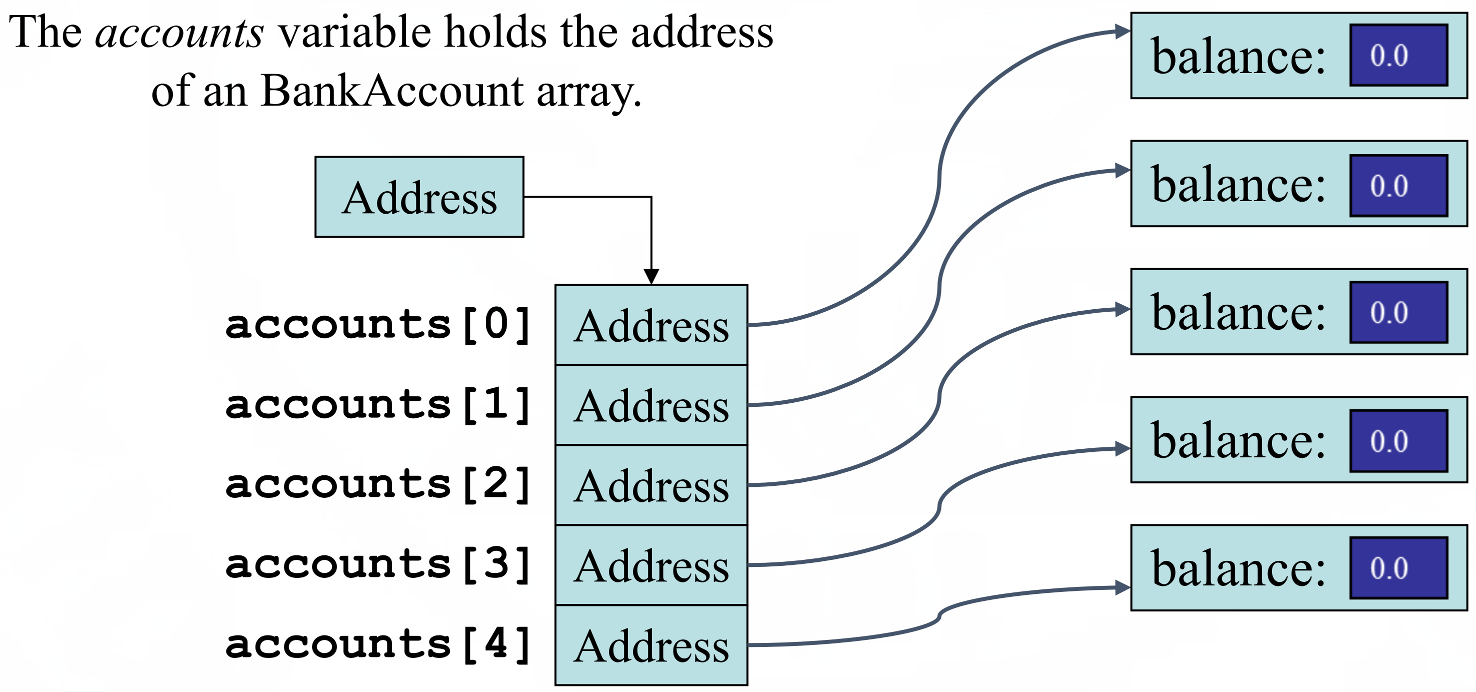
The array is an array of references to `BankAccount` objects.

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



ObjectArray.java

```java
import java.util.Scanner; // Needed for the Scanner class

/**
   This program works with an array of three
   BankAccount objects.
*/

public class ObjectArray
{
   public static void main(String[] args)
   {
      final int NUM_ACCOUNTS = 3;  // Number of accounts

      // Create an array that can reference
      // BankAccount objects.
      BankAccount[] accounts = new BankAccount[NUM_ACCOUNTS];

      // Create objects for the array.
      createAccounts(accounts);

      // Display the balances of each account.
      System.out.println("Here are the balances " +
                         "for each account:");

      for (int index = 0; index < accounts.length; index++)
      {
         System.out.println("Account " + (index + 1) +
                    ": $" + accounts[index].getBalance());
      }
   }

   /**
      The createAccounts method creates a BankAccount
      object for each element of an array. The user
      is asked for each account's balance.
      @param array The array to reference the accounts
   */

   private static void createAccounts(BankAccount[] array)
   {
      double balance;   // To hold an account balance

      // Create a Scanner object.
      Scanner keyboard = new Scanner(System.in);

      // Create the accounts.
      for (int index = 0; index < array.length; index++)
      {
         // Get the account's balance.
         System.out.print("Enter the balance for " +
                          "account " + (index + 1) + ": ");
         balance = keyboard.nextDouble();

         // Create the account.
         array[index] = new BankAccount(balance);
      }
   }
}
```

Program Output with Example Input Shown in Bold

```
Enter the balance for account 1: 2500.0 [Enter]
Enter the balance for account 2: 5000.0 [Enter]
Enter the balance for account 3: 1500.0 [Enter]
Here are the balances for each account:
Account 1: $2500.0
Account 2: $5000.0
Account 3: $1500.0
```

The Sequential Search Algorithm

- A search algorithm is a method of locating a specific item in a larger collection of data.
- The *sequential search algorithm* uses a loop to:
  - sequentially step through an array,
  - compare each element with the search value, and
  - stop when
    - the value is found or
    - the end of the array is encountered.
- See example: SearchArray.java

SearchArray.java

```java
/**
   This program sequentially searches an
   int array for a specified value.
*/

public class SearchArray
{
   public static void main(String[] args)
   {
      int[] tests = { 87, 75, 98, 100, 82 };
      int results;

      // Search the array for the value 100.
      results = sequentialSearch(tests, 100);

      // Determine whether 100 was found and
      // display an appropriate message.
      if (results == -1)
      {
         System.out.println("You did not " +
                     "earn 100 on any test.");
      }
      else
      {
         System.out.println("You earned 100 " +
                     "on test " + (results + 1));
      }
   }

   /**
      The sequentialSearch method searches an array for
      a value.
      @param array The array to search.
      @param value The value to search for.
      @return The subscript of the value if found in the
              array, otherwise -1.
   */

   public static int sequentialSearch(int[] array,
                                      int value)
   {
      int index;         // Loop control variable
      int element;       // Element the value is found at
      boolean found;     // Flag indicating search results

      // Element 0 is the starting point of the search.
      index = 0;

      // Store the default values element and found.
      element = -1;
      found = false;

      // Search the array.
      while (!found && index < array.length)
      {
         if (array[index] == value)
         {
            found = true;
            element = index;
         }
         index++;
      }

      return element;
   }
}
```

Program Output

```
You earned 100 on test 4
```

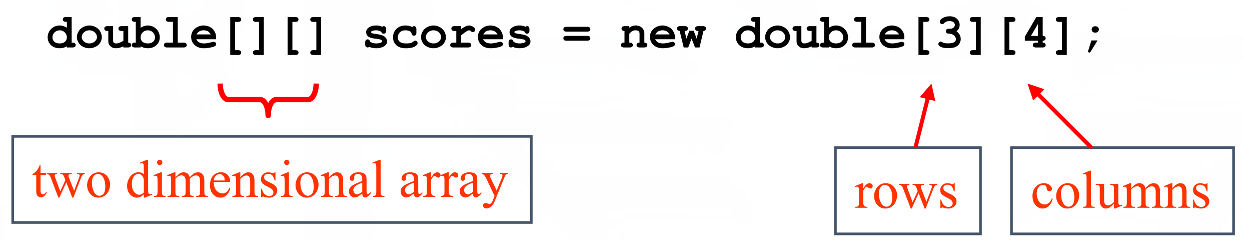
Two-Dimensional Arrays (1 of 2)

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

	column 0	column 1	column 2	column 3
row 0				
row 1				
row 2				
row 3				

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];
```

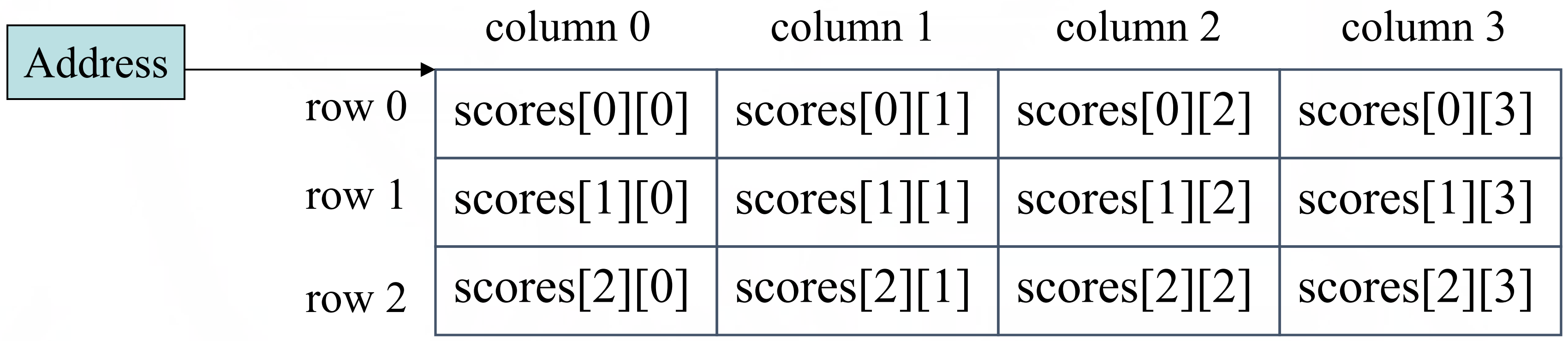
- The two sets of brackets in the data type indicate that the scores variable will reference a two-dimensional array.
- Notice that each size declarator is enclosed in its own set of brackets.

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.

Accessing Two-Dimensional Array Elements (2 of 5)

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

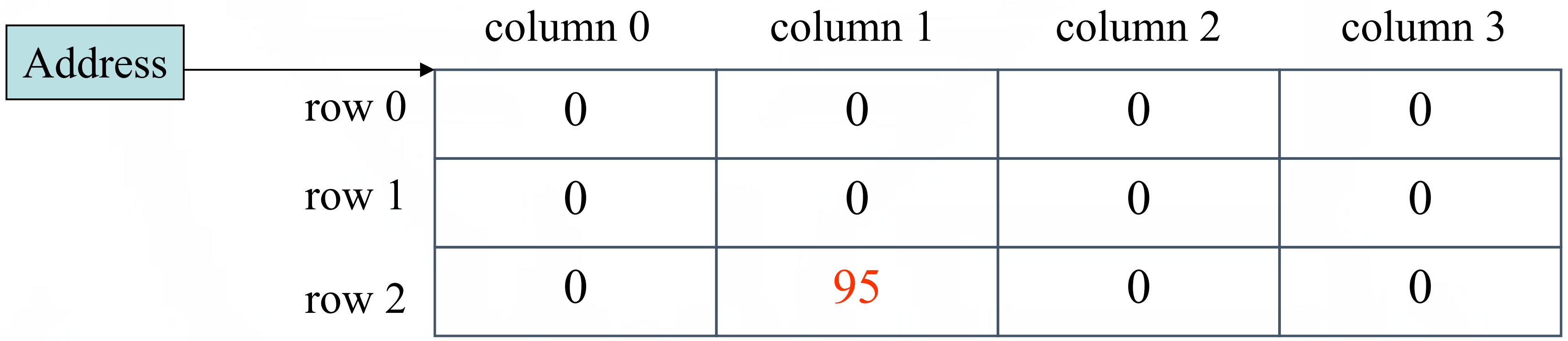


Accessing Two-Dimensional Array Elements (3 of 5)

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

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

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



	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

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:

```
for (int row = 0; row < 3; row++)
{
  for (int col = 0; col < 4; col++)
 {
    System.out.print("Enter a score: ");
    scores[row][col] = keyboard.nextDouble();
  }
}
```

Number of rows, not the largest subscript

Number of columns, not the largest subscript

`keyboard` references a `Scanner` object

Accessing Two-Dimensional Array Elements (5 of 5)

- To print out the `scores` array:

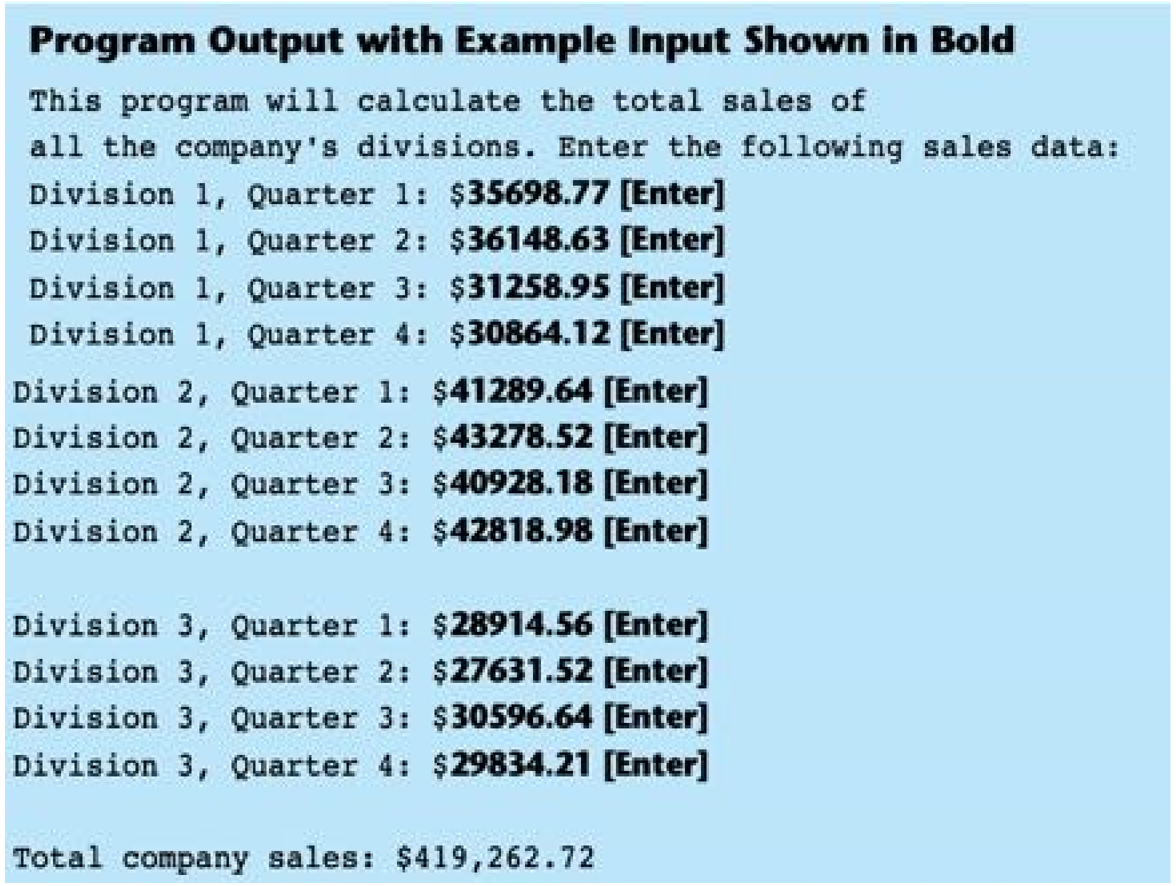
```
for (int row = 0; row < 3; row++)
{
   for (int col = 0; col < 4; col++)
    {
      System.out.println(scores[row][col]);
   }
}
```

- See example: CorpSales.java

CorpSales.java

```java
import java.util.Scanner;

/**
   This program demonstrates a two-dimensional array.
*/

public class CorpSales
{
   public static void main(String[] args)
   {
      final int DIVS = 3; // Three divisions in the company
      final int QTRS = 4; // Four quarters
      double totalSales = 0.0; // Accumulator

      // Create an array to hold the sales for each
      // division, for each quarter.
      double[][] sales = new double[DIVS][QTRS];

      // Create a Scanner object for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      // Display an introduction.
      System.out.println("This program will calculate the " +
                         "total sales of");
      System.out.println("all the company's divisions. " +
                         "Enter the following sales data:");

      // Nested loops to fill the array with quarterly
      // sales figures for each division.
      for (int div = 0; div < DIVS; div++)
      {
         for (int qtr = 0; qtr < QTRS; qtr++)
         {
            System.out.printf("Division %d, Quarter %d: $",
                              (div + 1), (qtr + 1));
            sales[div][qtr] = keyboard.nextDouble();
         }
         System.out.println();   // Print blank line.
      }

      // Nested loops to add all the elements of the array.
      for (int div = 0; div < DIVS; div++)
      {
         for (int qtr = 0; qtr < QTRS; qtr++)
         {
            totalSales += sales[div][qtr];
         }
      }

      // Display the total sales.
      System.out.printf("Total company sales: $%,.2f\n",
                        totalSales);
   }
}
```

CorpSales.java



Program Output with Example Input Shown in Bold

```
This program will calculate the total sales of
all the company's divisions. Enter the following sales data:
Division 1, Quarter 1: $**35698.77 [Enter]**
Division 1, Quarter 2: $**36148.63 [Enter]**
Division 1, Quarter 3: $**31258.95 [Enter]**
Division 1, Quarter 4: $**30864.12 [Enter]**
Division 2, Quarter 1: $**41289.64 [Enter]**
Division 2, Quarter 2: $**43278.52 [Enter]**
Division 2, Quarter 3: $**40928.18 [Enter]**
Division 2, Quarter 4: $**42818.98 [Enter]**

Division 3, Quarter 1: $**28914.56 [Enter]**
Division 3, Quarter 2: $**27631.52 [Enter]**
Division 3, Quarter 3: $**30596.64 [Enter]**
Division 3, Quarter 4: $**29834.21 [Enter]**

Total company sales: $419,262.72
```

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}

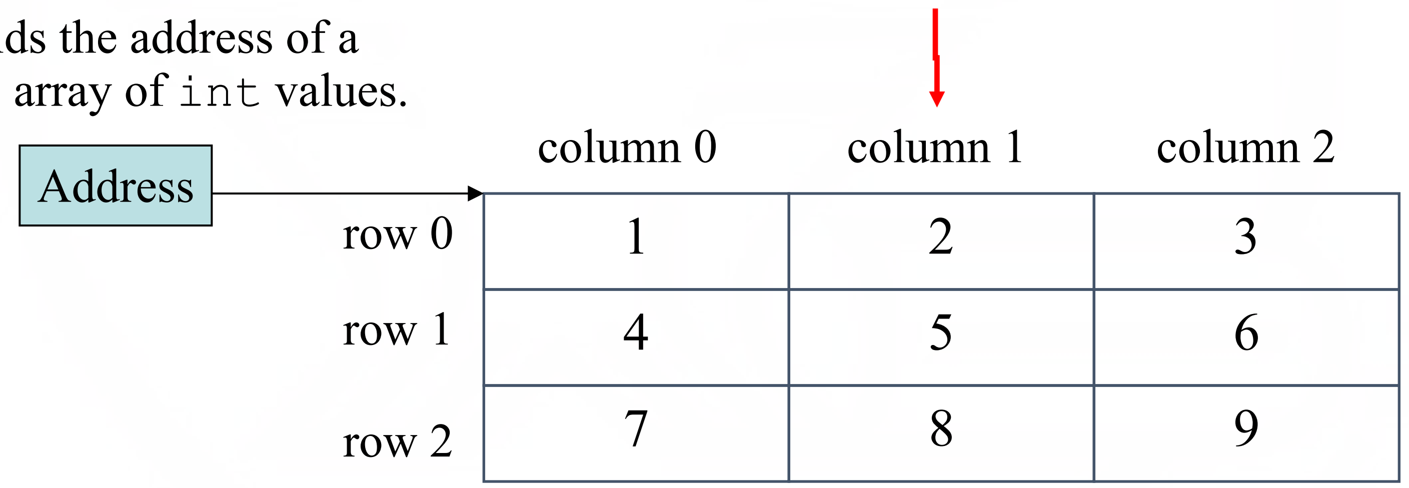
- Declares an array with three rows and three columns.

Initializing a Two-Dimensional Array (2 of 2)

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

produces:

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



	column 0	column 1	column 2
row 0	1	2	3
row 1	4	5	6
row 2	7	8	9

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.

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]);
}
```

- See example: Lengths.java

Number of rows

Number of columns in this row.

The array can have variable length rows.

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);
```

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);
}
```

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);
}
```

Passing and Returning Two-Dimensional Array References

- There is no difference between passing a single or two-dimensional array as an argument to a method.
- The method must accept a two-dimensional array as a parameter.
- See example: Pass2Darray.java

```java
/**
   This program demonstrates methods that accept
   a two-dimensional array as an argument.
*/

public class Pass2Darray
{
   public static void main(String[] args)
   {
      int[][] numbers = { { 1,  2,  3,  4 },
                          { 5,  6,  7,  8 },
                          { 9, 10, 11, 12 } };

      // Display the contents of the array.
      System.out.println("Here are the values " +
                          " in the array.");
      showArray(numbers);

      // Display the sum of the array's values.
      System.out.println("The sum of the values " +
                          "is " + arraySum(numbers));
   }

   /**
      The showArray method displays the contents
      of a two-dimensional int array.
      @param array The array to display.
   */

   private static void showArray(int[][] array)
   {
      for (int row = 0; row < array.length; row++)
      {
         for (int col = 0; col < array[row].length; col++)
            System.out.print(array[row][col] + " ");
         System.out.println();
      }
   }

   /**
      The arraySum method returns the sum of the
      values in a two-dimensional int array.
      @param array The array to sum.
      @return The sum of the array elements.
   */

   private static int arraySum(int[][] array)
   {
      int total = 0;    // Accumulator

      for (int row = 0; row < array.length; row++)
      {
         for (int col = 0; col < array[row].length; col++)
            total += array[row][col];
      }


      return total;
   }
}
```

Program Output

```
Here are the values in the array.
1 2 3 4
5 6 7 8
9 10 11 12
The sum of the values is 78
```

Ragged Arrays

- When the rows of a two-dimensional array are of different lengths, the array is known as a *ragged array*.
- You can create a ragged array by creating a two-dimensional array with a specific number of rows, but no columns.

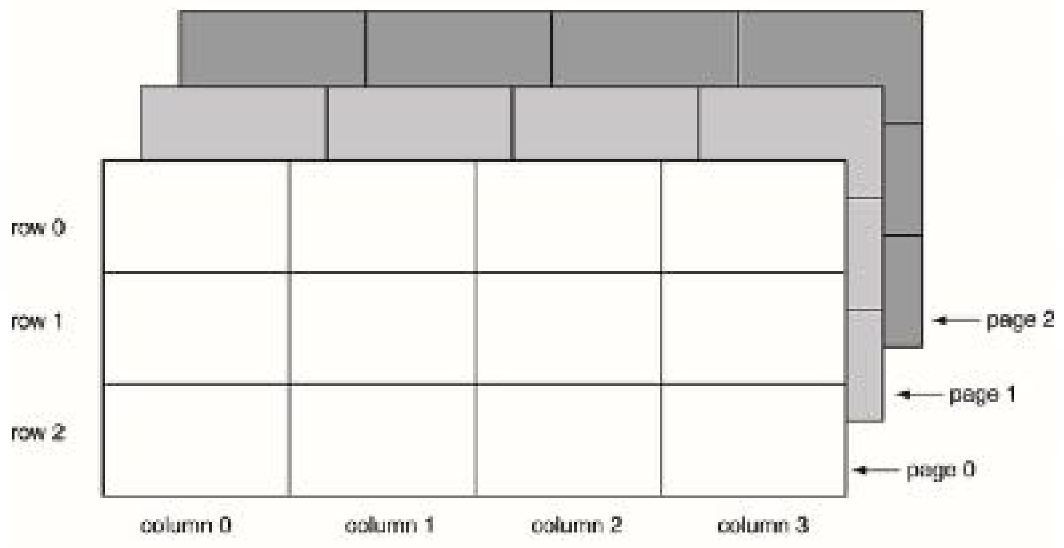
```
int [][] ragged = new int [4][];
```

- Then create the individual rows.

```
ragged[0] = new int [3];
ragged[1] = new int [4];
ragged[2] = new int [5];
ragged[3] = new int [6];
```

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.

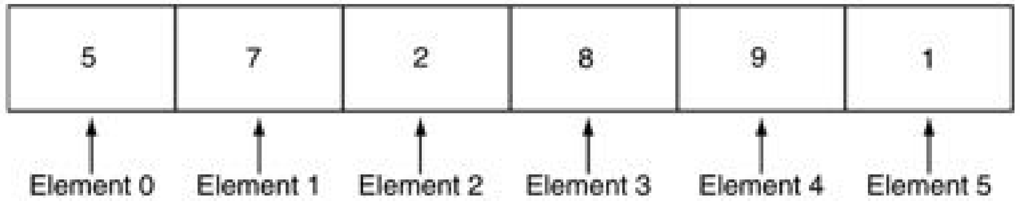


Selection Sort

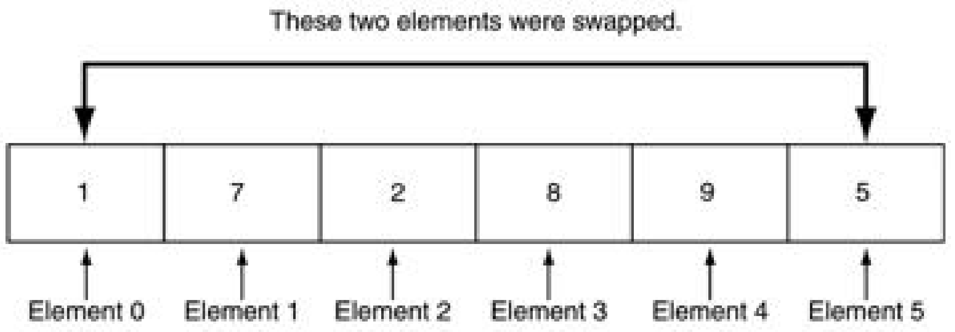
- In a selection sort:
  - The smallest value in the array is located and moved to element 0.
  - Then the next smallest value is located and moved to element 1.
  - This process continues until all of the elements have been placed in their proper order.
  - See example: SelectionSortDemo.java

Selection Sort

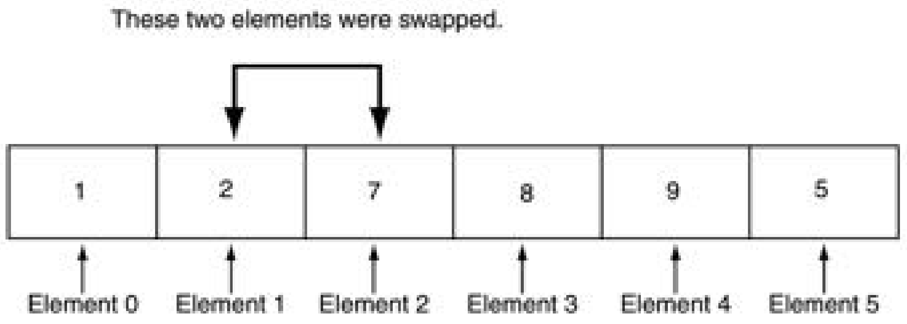
1



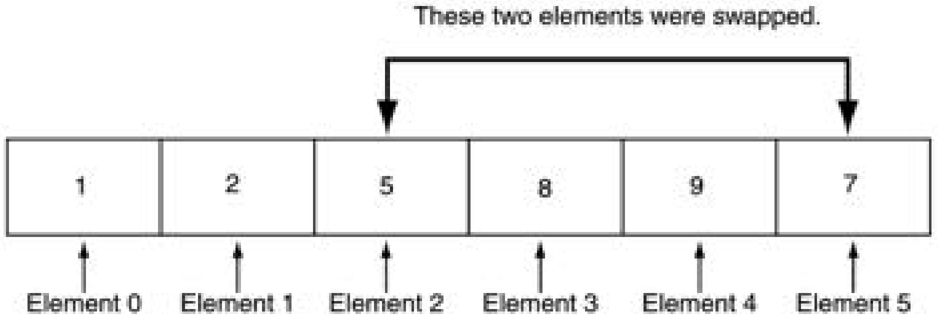
2



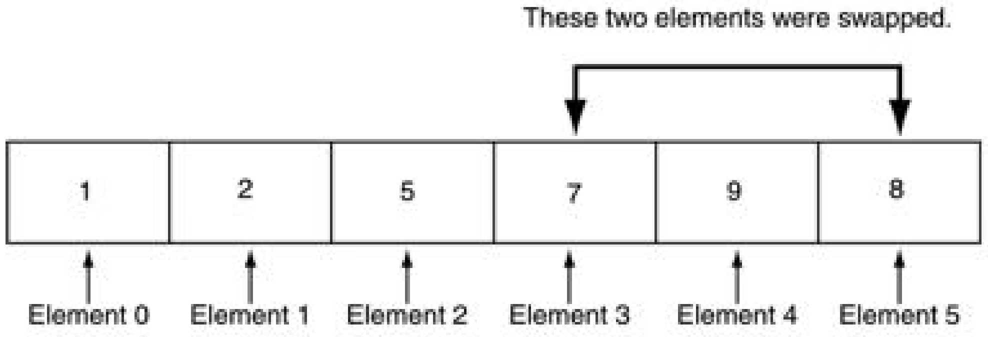
3



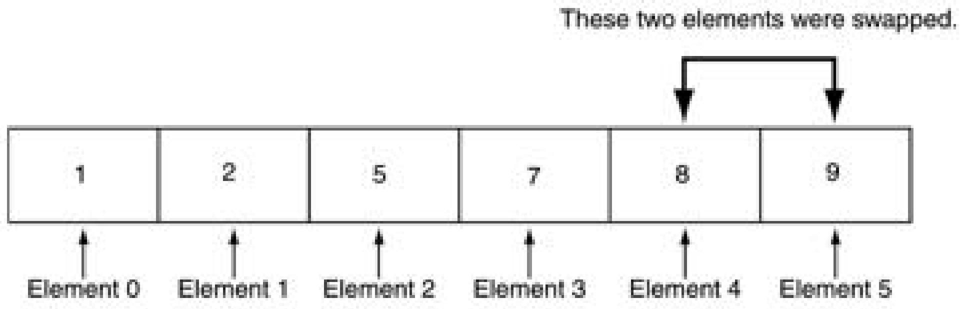
4



5



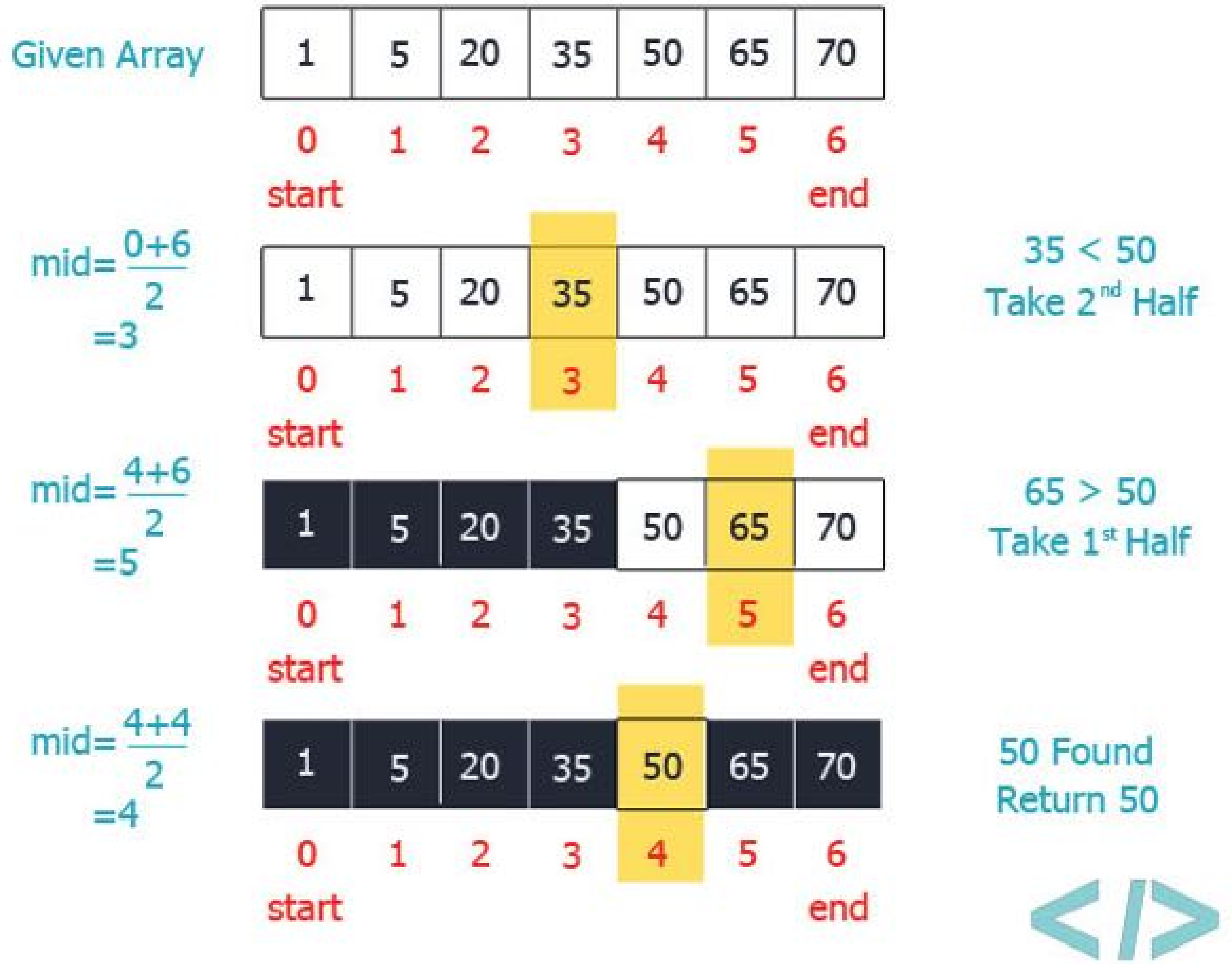
6



Binary Search

- A binary search:
  - requires an array sorted in ascending order.
  - starts with the element in the middle of the array.
  - If that element is the desired value, the search is over.
  - Otherwise, the value in the middle element is either greater or less than the desired value
  - If it is greater than the desired value, search in the first half of the array.
  - Otherwise, search the last half of the array.
  - Repeat as needed while adjusting start and end points of the search.
- See example: BinarySearchDemo.java

Binary Search for 50 in 7 elements Array



Command-Line Arguments (1 of 2)

- A Java program can receive arguments from the operating system command-line.
- The `main` method has a header that looks like this:

```
public static void main(String[] args)
```

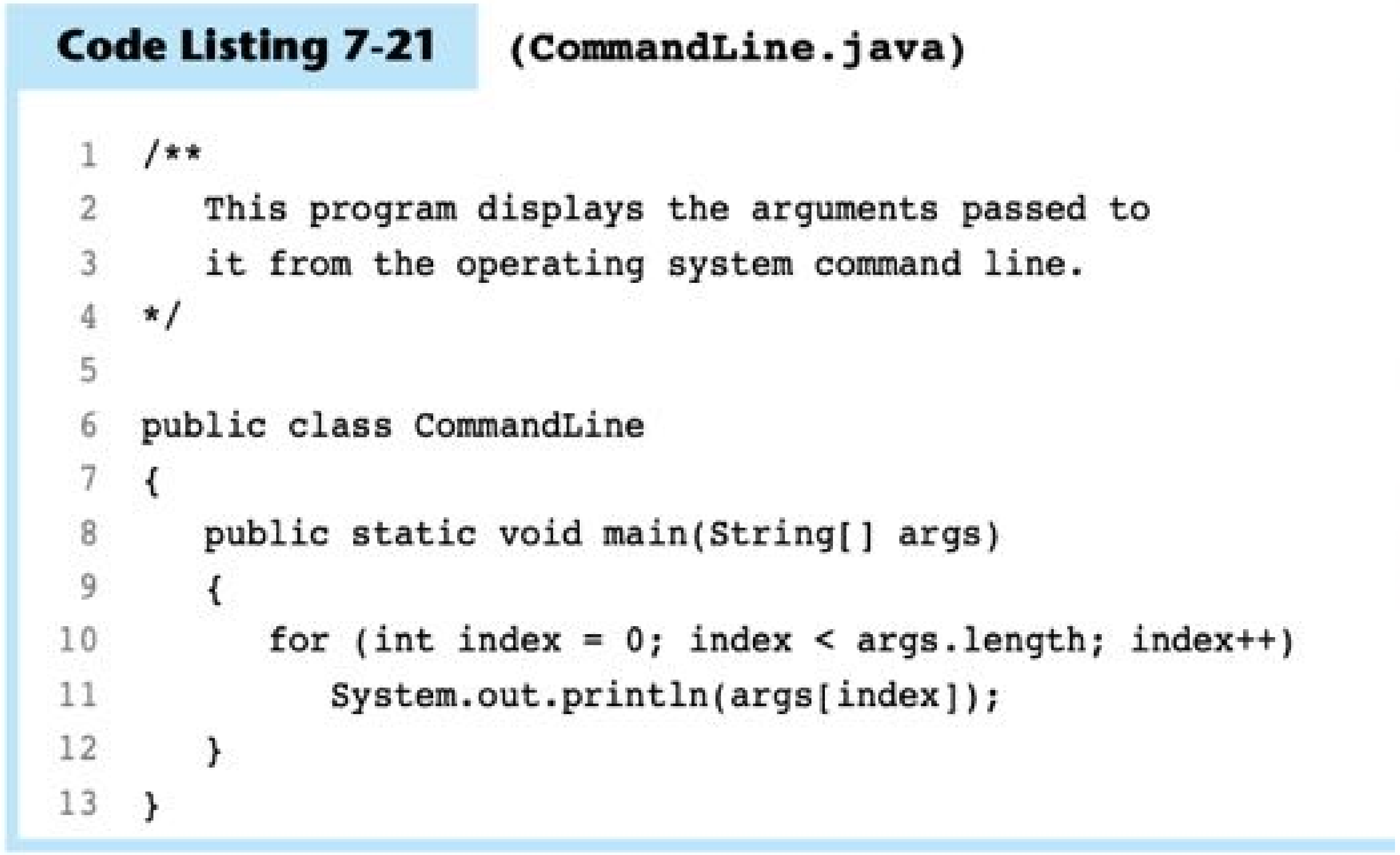
- The `main` method receives a `String` array as a parameter.
- The array that is passed into the `args` parameter comes from the operating system command-line.

Command-Line Arguments (2 of 2)

- To run the example:

```
java CommandLine How does this work?
   args[0] is assigned "How"
   args[0] is assigned "does"
   args[0] is assigned "this"
   args[0] is assigned "work?"
```

- Example: CommandLine.java
- It is not required that the name of `main`’s parameter array be `args`.



Code Listing 7-21 **(CommandLine.java)**

```java
/**
   This program displays the arguments passed to
   it from the operating system command line.
*/

public class CommandLine
{
   public static void main(String[] args)
   {
      for (int index = 0; index < args.length; index++)
         System.out.println(args[index]);
   }
}
```

If this program is compiled and then executed with the following command:

```
java CommandLine How does this work?
```

its output will be as follows:

```
How
does
this
work?
```

Variable-Length Argument Lists

- Special type parameter – vararg…
  - Vararg parameters are actually arrays
  - Examples: VarArgsDemo1.java, VarargsDemo2.java

```
public static int sum(int... numbers)
{
    int total = 0; // Accumulator
    // Add all the values in the numbers array.
    for (int val : numbers)
     total += val;
    // Return the total.
    return total;
}
```

The `ArrayList` Class

- Similar to an array, an `ArrayList` allows object storage

- Unlike an array, an `ArrayList` object:
  - Automatically expands when a new item is added
  - Automatically shrinks when items are removed

- Requires:

```
    import java.util.ArrayList;
```

Creating an `ArrayList`

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

Notice the word `String` written inside angled brackets <>

This specifies that the `ArrayList` can hold `String` objects.

If we try to store any other type of object in this `ArrayList`, an error will occur.

Using an `ArrayList` (1 of 8)

- To populate the `ArrayList`, use the `add` method:
  - `nameList.add("James");`
  - `nameList.add("Catherine");`

- To get the current size, call the `size` method
  - `nameList.size();   // returns 2`

Using an `ArrayList` (2 of 8)

- To access items in an `ArrayList`, use the `get` method

  ```
  nameList.get(1);
  ```

  In this statement 1 is the index of the item to get.

- Example: ArrayListDemo1.java

Using an `ArrayList` (3 of 8)

- The `ArrayList` class's `toString` method returns a string representing all items in the `ArrayList`

```
System.out.println(nameList);
```

This statement yields :

```
[ James, Catherine ]
```

- The `ArrayList` class's `remove` method removes designated item from the `ArrayList`

```
nameList.remove(1);
```

This statement removes the second item.

- See example: ArrayListDemo3.java

Using an `ArrayList` (4 of 8)

- The `ArrayList` class's `add` method with one argument adds new items to the end of the `ArrayList`
- To insert items at a location of choice, use the `add` method with two arguments:

  ```
  nameList.add(1, "Mary");
  ```

  This statement inserts the `String` "Mary" at index 1
- To replace an existing item, use the `set` method:

  ```
  nameList.set(1, "Becky");
  ```

  This statement replaces “Mary” with “Becky”
- See example: ArrayListDemo5.java

Using an `ArrayList` (5 of 8)

- An `ArrayList` has a capacity, which is the number of items it can hold without increasing its size.
- The default capacity of an `ArrayList` is 10 items.
- To designate a different capacity, use a parameterized constructor:

```
ArrayList<String> list = new ArrayList<String>(100);
```

Using an `ArrayList` (6 of 8)

- You can store any type of *object* in an `ArrayList`

```
ArrayList<BankAccount> accountList =
                new ArrayList<BankAccount>();
```

This creates an `ArrayList` that can hold `BankAccount` objects.

Using an `ArrayList` (7 of 8)

```
// Create an ArrayList to hold BankAccount objects.
ArrayList<BankAccount> list = new ArrayList<BankAccount>();

// Add three BankAccount objects to the ArrayList.
list.add(new BankAccount(100.0));
list.add(new BankAccount(500.0));
list.add(new BankAccount(1500.0));

// Display each item.
for (int index = 0; index < list.size(); index++)
{
   BankAccount account = list.get(index);
   System.out.println("Account at index " + index +
                "\nBalance: " + account.getBalance());
}
```

See: ArrayListDemo6.java

Using an `ArrayList` (8 of 8)

- The diamond operator
  - Beginning in Java 7, you can use the <> operator for simpler `ArrayList` declarations:

No need to specify the data type here.

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

Java infers the type of the `ArrayList` object from the variable declaration.

рахмат
danke
謝謝
ngiyabonga
شكراً جزيلاً
spas
teşekkür ederim
Баярлалаа
mersi
kia ora
barka
welalin
tack
mahalo
tapadh leat
спасибо
faafetai lava
vinaka
blagodaram
dank je
misaotra
matondo
paldies
grazzi
спасибі
kiitos
dankie
nanni
nandri
dhanyavad
mauruuru
köszönöm
thank you
gracias
хвала
asante
manana
obrigada
tenki
bedankt
bayarlalaa
gracie
hvala
aciu
dankon
akun
djiere dieuf
tau
mochchakkeram
murakoze
enkosi
дякую
mamnun
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
taiku
go raibh maith agat
chokrane
sobodi
dekuji
sukriya
kop khun krap
sagolun
obrigado
mesi
didi madloba
ありがとう
najis tuke
tanemirt
grazie
arigatô
takk
dakujem
trugarez
kam sah hamnida
terima kasih
rahmat
rahmet
ευχαριστώ
diolch
dhanyavadagalu
shukriya
mercé
мерси
তোমাকে ধন্যবাদ
감사합니다
xiexie
merci