جامعة الجلالة
GALALA UNIVERSITY

CSE110 Principles of Programming

Lecture 7: Arrays and the ArrayList

Professor Shaker El-Sappagh

Shaker.elsappagh@gu.edu.eg

Fall 2023

Chapter Topics

Chapter 7 discusses the following main topics:

- Introduction to Arrays
- Processing Array Contents
- Passing Arrays as Arguments to Methods
- Some Useful Array Algorithms and Operations
- Returning Arrays from Methods
- String Arrays
- Arrays of Objects
- The Sequential Search Algorithm
- Parallel Arrays
- Two-Dimensional Arrays
- Arrays with Three or More Dimensions
- The Selection Sort and the Binary Search
- Command-Line Arguments
- The `ArrayList` Class

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.

Creating Arrays

- 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];
```

0	0	0	0	0	0
index 0	index 1	index 2	index 3	index 4	index 5

Array element values are initialized to 0.

Array indexes always start at 0.

Creating Arrays

- 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

- 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.

Accessing the Elements of an Array

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

- 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"
```

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.

```java
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]);
}
```

Array subscripts can be accessed using variables (such as for loop counters).

```
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
```

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.
- 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*.

Invalid Subscript

```
public class InvalidSubscript
{
   public static void main(String[] args)
   {
      int[] values = new int[3];

      System.out.println("I will attempt to store four " +
                         "numbers in a three-element array.");

      for (int index = 0; index < 4; index++)
      {
         System.out.println("Now processing element " + index);
         values[index] = 10;
      }
   }
}
```

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)
```

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`.

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.

Array Initialization

```
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.
```

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

- 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
```

```java
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

- 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

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

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

The Enhanced `for` Loop

Example:

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

Array Size

- 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

- 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];
```

```java
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
```

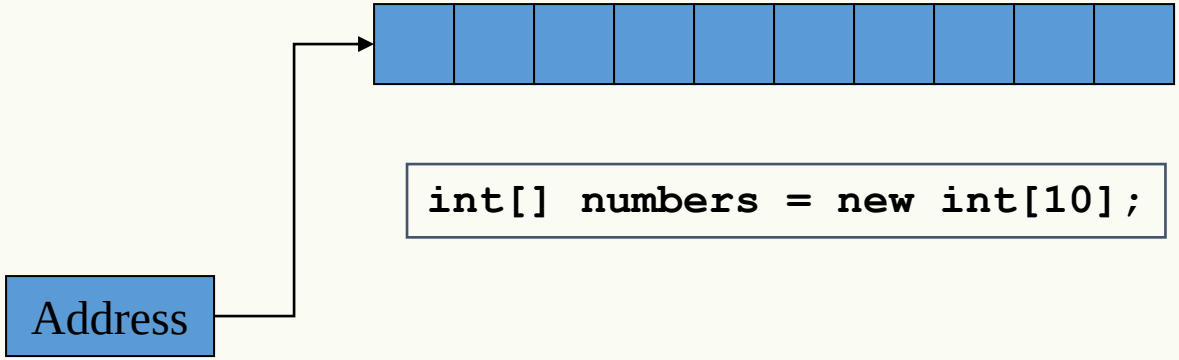
Reassigning Array References

- 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

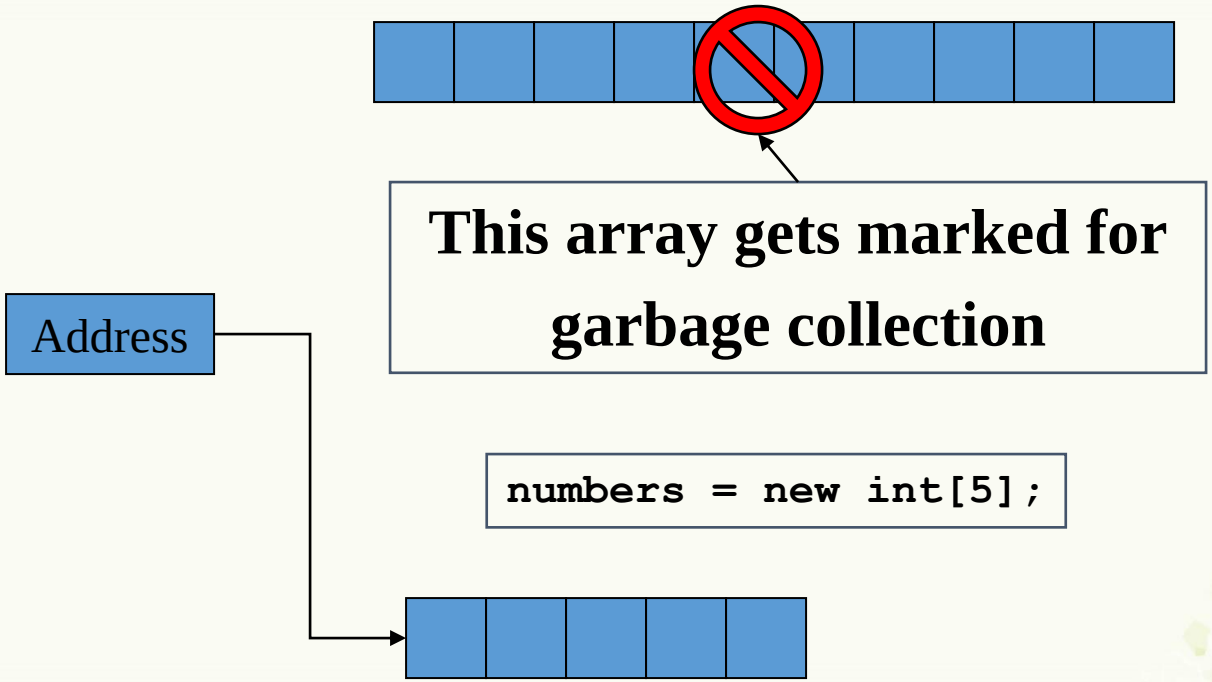



```
int[] numbers = new int[10];
```

The `numbers` variable holds the address of an `int` array.

Reassigning Array References

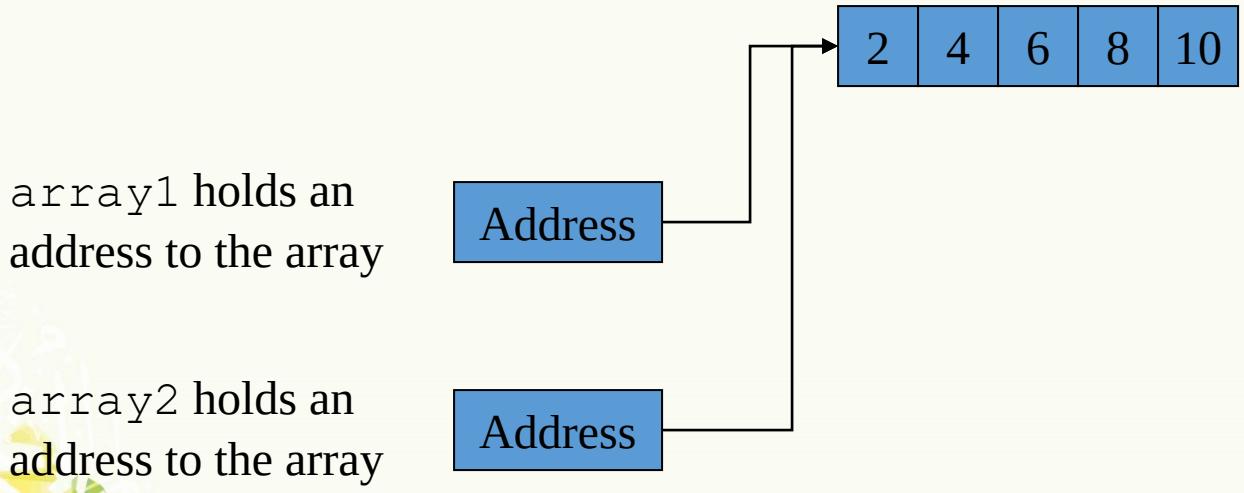
The `numbers` variable holds the address of an `int` array.



Copying Arrays

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

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



```java
public class SameArray
{
   public static void main(String[] args)
   {
      int[] array1 = { 2, 4, 6, 8, 10 };
      int[] array2 = array1;

      // Change one of the elements using array1.
      array1[0] = 200;

      // Change one of the elements using array2.
      array2[4] = 1000;

      // Display all the elements using array1
      System.out.println("The contents of array1:");
      for (int value : array1)
         System.out.print(value + " ");
      System.out.println();

      // Display all the elements using array2
      System.out.println("The contents of array2:");
      for (int value : array2)
         System.out.print(value + " ");
      System.out.println();
   }
}
```

Program Output

```
The contents of array1:
200 4 6 8 1000
The contents of array2:
200 4 6 8 1000
```

Copying Arrays

- 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.

- More often you will want to write methods to process array data by passing the entire array, not just one element at a time.

```java
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 to Methods (Cont.)

- Pass-by-value (also called call-by-value)
  - A copy of the argument's *value is passed to the called method.*
  - The called method works exclusively with the copy.
  - Changes to the called method's copy do not affect the original variable's value in the caller.
- Pass-by-reference (also called call-by-reference)
  - The called method can access the argument's value in the caller directly and modify that data, if necessary.
  - Improves performance by eliminating the need to copy possibly large amounts of data.

Passing Arrays to Methods (Cont.)

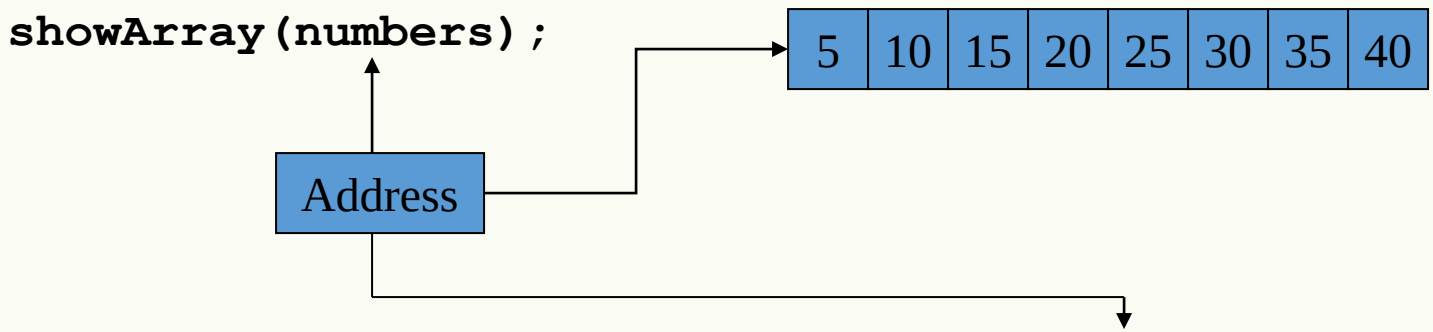
- All arguments in Java are passed by value.
- A method call can pass two types of values to a method
  - Copies of primitive values
  - Copies of references to objects
- Objects cannot be passed to methods.
- If a method modifies a reference-type parameter so that it refers to another object, only the parameter refers to the new object
  - The reference stored in the caller's variable still refers to the original object.
- Although an object's reference is passed by value, a method can still interact with the referenced object by calling its `public` methods using the copy of the object's reference.
  - The parameter in the called method and the argument in the calling method refer to the same object in memory.

Passing Arrays to Methods

- To pass an array argument to a method, specify the name of the array without any brackets.
  - Since every array object "knows" its own length, we need not pass the array length as an additional argument.
- To receive an array, the method's parameter list must specify an array parameter.
- When an argument to a method is an entire array or an individual array element of a reference type, the called method receives a copy of the reference.
- When an argument to a method is an individual array element of a primitive type, the called method receives a copy of the element's value.
  - Such primitive values are called scalars or scalar quantities.

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 1

```
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);
}
```

```
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();
   }
}
```

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

   }
}
```

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
```

Example 2

```
public class PassArray
{
   // main creates array and calls modifyArray and modifyElement
   public static void main( String[] args )
   {
      int[] array = { 1, 2, 3, 4, 5 };

      System.out.println(
         "Effects of passing reference to entire array:\n" +
         "The values of the original array are:" );

      // output original array elements
      for ( int value : array )
         System.out.printf( "   %d", value );

      modifyArray( array ); // pass array reference
      System.out.println( "\n\nThe values of the modified array are:" );

      // output modified array elements
      for ( int value : array )
         System.out.printf( "   %d", value );
```

Passes the reference to `array` into method `modifyArray`

```java
      System.out.printf(
         "\n\nEffects of passing array element value:\n" +
         "array[3] before modifyElement: %d\n", array[ 3 ] );

      modifyElement( array[ 3 ] ); // attempt to modify array[ 3 ]
      System.out.printf(
         "array[3] after modifyElement: %d\n", array[ 3 ] );
   } // end main

   // multiply each element of an array by 2
   public static void modifyArray( int[] array2 )
   {
      for ( int counter = 0; counter < array2.length; counter++ )
         array2[ counter ] *= 2;
   } // end method modifyArray

   // multiply argument by 2
   public static void modifyElement( int element )
   {
      element *= 2;
      System.out.printf(
         "Value of element in modifyElement: %d\n", element );
   } // end method modifyElement
} // end class PassArray
```

Passes a copy of `array[3]`'s `int` value into `modifyElement`

Method receives copy of an array's reference, which gives the method direct access to the original array in memory

Method receives copy of an `int` value; the method cannot modify the original `int` value in `main`

```
Effects of passing reference to entire array:
The values of the original array are:
   1   2   3   4   5

The values of the modified array are:
   2   4   6   8   10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 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

- 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];
}
```

Useful Array Operations

- 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;
```

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

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.

```java
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
```

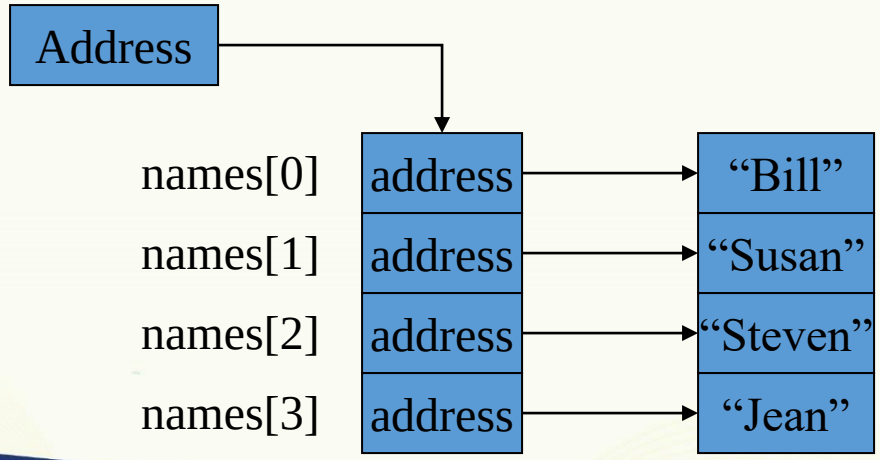
`String` Arrays

- 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.



```java
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

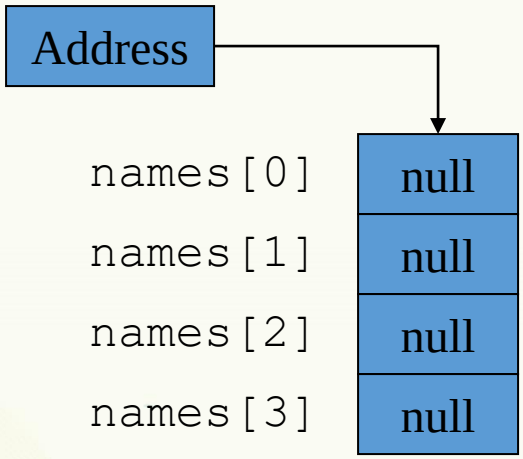
```
January has 31 days.
February has 28 days.
March has 31 days.
April has 30 days.
May has 31 days.
June has 30 days.
July has 31 days.
August has 31 days.
September has 30 days.
October has 31 days.
November has 30 days.
December has 31 days.
```

`String` Arrays

- 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.

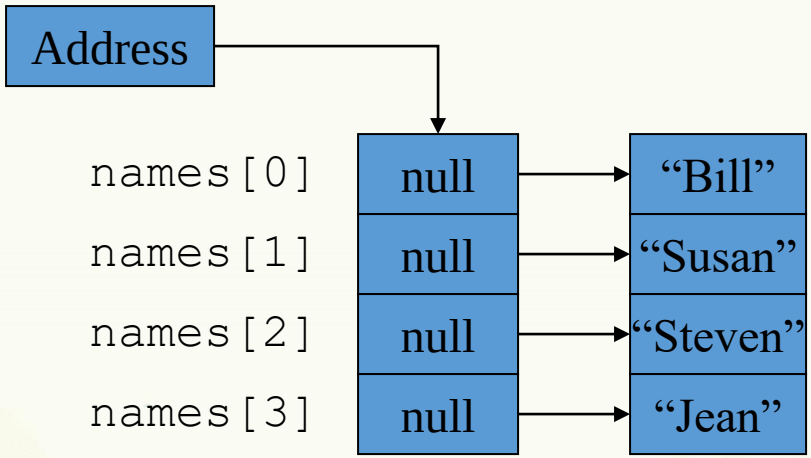


`String` Arrays

- 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.



Calling `String` Methods On Array Elements

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

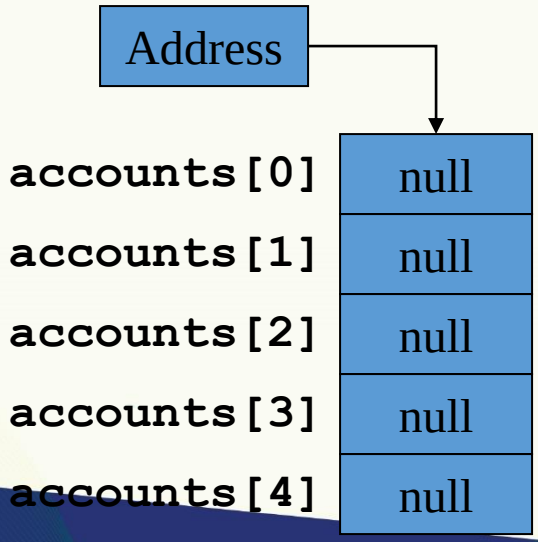
- An array's `length` is a **field**
  - You <u>do not</u> write a set of parentheses after its name.
- A `String`'s `length` is a **method**
  - You <u>do</u> write the parentheses after the name of the `String` class's `length` method.

Arrays of Objects

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

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

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



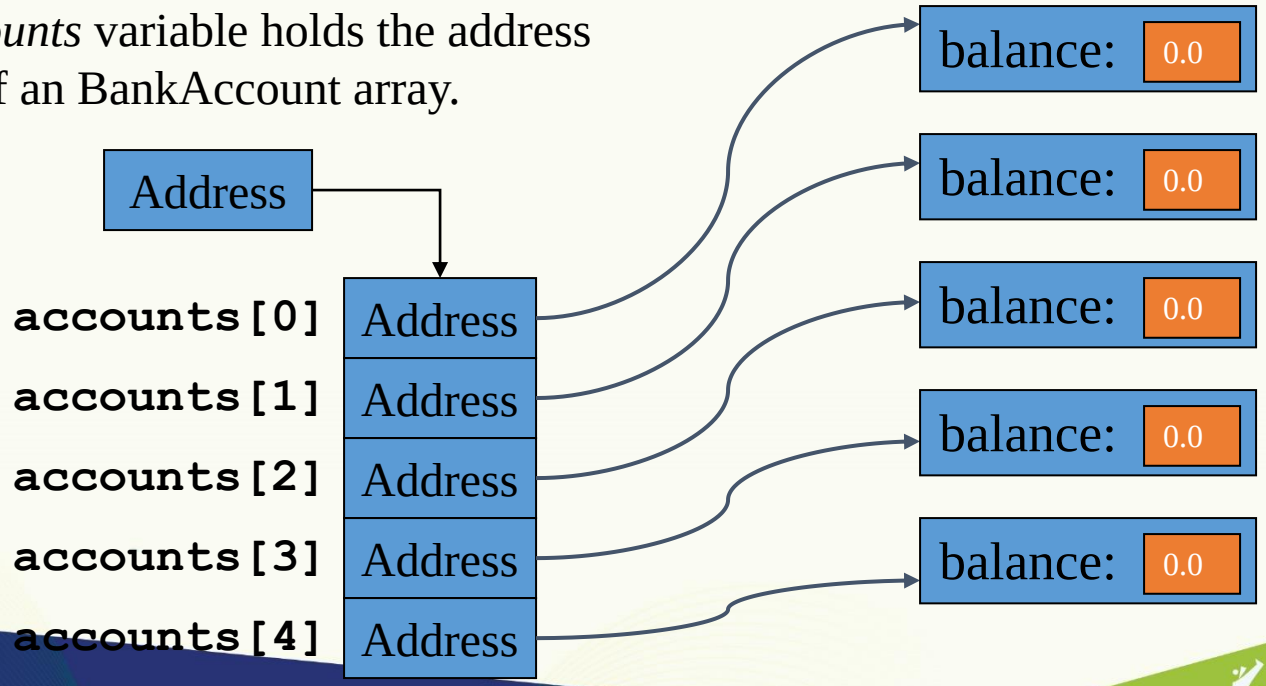
The array is an array of references to `BankAccount` objects.

Arrays of Objects

- Each element needs to be initialized.

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

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



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.

```java
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));
      }
   }
```

Program Output

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

```java
   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;
   }
}
```

NOTE: The reason −1 is returned when the search value is not found in the array is because −1 is not a valid subscript.

Two-Dimensional Arrays

- 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

- 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];
```

two dimensional array

rows

columns

- 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

- 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

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

Address

	column 0	column 1	column 2	column 3
row 0	scores[0][0]	scores[0][1]	scores[0][2]	scores[0][3]
row 1	scores[1][0]	scores[1][1]	scores[1][2]	scores[1][3]
row 2	scores[2][0]	scores[2][1]	scores[2][2]	scores[2][3]

Accessing Two-Dimensional Array Elements

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.

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

Accessing Two-Dimensional Array Elements

- 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

- 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]);
  }
}
```

```
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
 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:");
```

```
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);
 }
}
```

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

- 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

```
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.

Address

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

The `length` Field

- 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

- 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.

The array can have variable length rows.

```java
public class Lengths
{
   public static void main(String[] args)
   {
      // Declare a 2D array with 3 rows
      // and 4 columns.

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

      // Display the number of rows.
      System.out.println("The number of " +
                 "rows is " + numbers.length);

      // Display the number of columns in each row.
      for (int index = 0; index < numbers.length; index++)
      {
         System.out.println("The number of " +
               "columns in row " + index + " is " +
               numbers[index].length);
      }
   }
}
```

Program Output

```
The number of rows is 3
The number of columns in row 0 is 4
The number of columns in row 1 is 4
The number of columns in row 2 is 4
```

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.

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

   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));
}
```

```java
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.

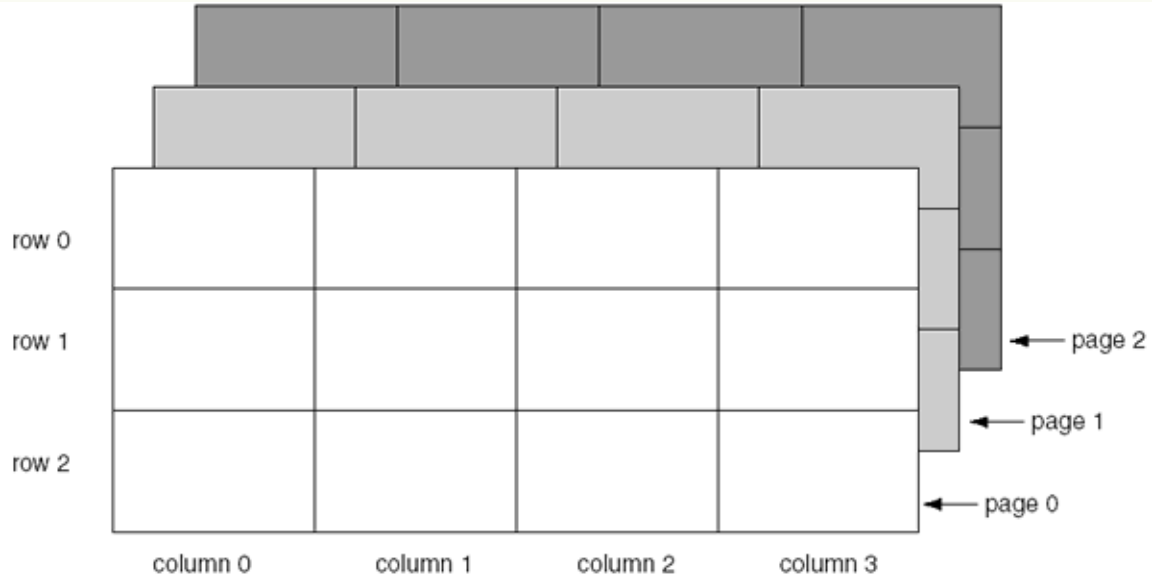
```
ragged[0] = new int [3]; // Row 0 has 3 columns.
ragged[1] = new int [4]; // Row 1 has 4 columns.
ragged[2] = new int [5]; // Row 2 has 5 columns.
ragged[3] = new int [6]; // Row 3 has 6 columns.
```

This code creates the four rows. Row 0 has three columns, row 1 has four columns, row 2 has five columns, and row 3 has six columns. The following code displays the number of columns in each row:

```
for (int index = 0; index < ragged.length; index++)
{
   System.out.println("The number of columns " +
                      "in row " + index + " is " +
                      ragged[index].length);
}
```

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.



Command-Line Arguments

- 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

- 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?"
```

- It is not required that the name of `main`'s parameter array be `args`.

Example 1

```
/**
   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]);
   }
}
```

Example 2

```
public class InitArray
{
   public static void main( String[] args )
   {
      // check number of command-line arguments
      if ( args.length != 3 )
         System.out.println(
            "Error: Please re-enter the entire command, including\n" +
            "an array size, initial value and increment." );
      else
      {
         // get array size from first command-line argument
         int arrayLength = Integer.parseInt( args[ 0 ] );
         int[] array = new int[ arrayLength ]; // create array

         // get initial value and increment from command-line arguments
         int initialValue = Integer.parseInt( args[ 1 ] );
         int increment = Integer.parseInt( args[ 2 ] );
```

```java
         // calculate value for each array element
         for ( int counter = 0; counter < array.length; counter++ )
            array[ counter ] = initialValue + increment * counter;

         System.out.printf( "%s%8s\n", "Index", "Value" );

         // display array index and value
         for ( int counter = 0; counter < array.length; counter++ )
            System.out.printf( "%5d%8d\n", counter, array[ counter ] );
      } // end else
   } // end main
} // end class InitArray
```

```
java InitArray
Error: Please re-enter the entire command, including
an array size, initial value and increment.
```

```
java InitArray 5 0 4
Index   Value
    0       0
    1       4
    2       8
    3      12
    4      16
```

```
java InitArray 8 1 2
Index   Value
    0       1
    1       3
    2       5
    3       7
    4       9
    5      11
    6      13
    7      15
```

Variable-Length Argument Lists

- Special type parameter – vararg…
  - Vararg parameters are actually arrays

```
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`

- 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`

- 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.

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

/**
   This program demonstrates an ArrayList.
*/

public class ArrayListDemo1
{
   public static void main(String[] args)
   {
      // Create an ArrayList to hold some names.
      ArrayList<String> nameList = new ArrayList<String>();

      // Add some names to the ArrayList.
      nameList.add("James");
      nameList.add("Catherine");
      nameList.add("Bill");

      // Display the size of the ArrayList.
      System.out.println("The ArrayList has " +
                         nameList.size() +
                         " objects stored in it.");

      // Now display the items in nameList.
      for (int index = 0; index < nameList.size(); index++)
         System.out.println(nameList.get(index));
   }
}
```

Program Output

```
The ArrayList has 3 objects stored in it.
James
Catherine
Bill
```

Using an `ArrayList`

- 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.

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

/**
   This program demonstrates an ArrayList.
*/

public class ArrayListDemo3
{
   public static void main(String[] args)
   {
      // Create an ArrayList to hold some names.
      ArrayList<String> nameList = new ArrayList<String>();

      // Add some names to the ArrayList.
      nameList.add("James");
      nameList.add("Catherine");
      nameList.add("Bill");

      // Display the items in nameList and their indices.
      for (int index = 0; index < nameList.size(); index++)
      {
         System.out.println("Index: " + index + " Name: " +
                            nameList.get(index));
      }

      // Now remove the item at index 1.
      nameList.remove(1);

      System.out.println("The item at index 1 is removed. " +
                         "Here are the items now.");

      // Display the items in nameList and their indices.
```

```
        for (int index = 0; index < nameList.size(); index++)
        {
           System.out.println("Index: " + index + " Name: " +
                                nameList.get(index));
        }
    }
}
```

Program Output

```
Index: 0 Name: James
Index: 1 Name: Catherine
Index: 2 Name: Bill
The item at index 1 is removed. Here are the items now.
Index: 0 Name: James
Index: 1 Name: Bill
```

Using an `ArrayList`

- 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”

Using an `ArrayList`

- 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`

- 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`

```
// 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());
}
```

Using an `ArrayList`

- 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.

ArrayList

Method	Description
`add`	Adds an element to the end of the `ArrayList`.
`clear`	Removes all the elements from the `ArrayList`.
`contains`	Returns `true` if the `ArrayList` contains the specified element; otherwise, returns `false`.
`get`	Returns the element at the specified index.
`indexOf`	Returns the index of the first occurrence of the specified element in the `ArrayList`.
`remove`	Overloaded. Removes the first occurrence of the specified value or the element at the specified index.
`size`	Returns the number of elements stored in the `ArrayList`.
`trimToSize`	Trims the capacity of the `ArrayList` to current number of elements.

Class `Arrays`

- Arrays class
  - Provides `static` methods for common array manipulations.
- Methods include
  - sort for sorting an array (ascending order by default)
  - binarySearch for searching a sorted array
  - equals for comparing arrays
  - fill for placing values into an array.
- Methods are overloaded for primitive-type arrays and for arrays of objects.
- `System` class `static` arraycopy method
  - Copies contents of one array into another.

```java
import java.util.Arrays;

public class ArrayManipulations
{
   public static void main( String[] args )
   {
      // sort doubleArray into ascending order
      double[] doubleArray = { 8.4, 9.3, 0.2, 7.9, 3.4 };
      Arrays.sort( doubleArray );
      System.out.printf( "\ndoubleArray: " );

      for ( double value : doubleArray )
         System.out.printf( "%.1f ", value );

      // fill 10-element array with 7s
      int[] filledIntArray = new int[ 10 ];
      Arrays.fill( filledIntArray, 7 );
      displayArray( filledIntArray, "filledIntArray" );

```

```java
        // copy array intArray into array intArrayCopy
        int[] intArray = { 1, 2, 3, 4, 5, 6 };
        int[] intArrayCopy = new int[ intArray.length ];
        System.arraycopy( intArray, 0, intArrayCopy, 0, intArray.length );
        displayArray( intArray, "intArray" );
        displayArray( intArrayCopy, "intArrayCopy" );

        // compare intArray and intArrayCopy for equality
        boolean b = Arrays.equals( intArray, intArrayCopy );
        System.out.printf( "\n\nintArray %s intArrayCopy\n",
           ( b ? "==" : "!=" ) );

        // compare intArray and filledIntArray for equality
        b = Arrays.equals( intArray, filledIntArray );
        System.out.printf( "intArray %s filledIntArray\n",
           ( b ? "==" : "!=" ) );

        // search intArray for the value 5
        int location = Arrays.binarySearch( intArray, 5 );

        if ( location >= 0 )
           System.out.printf(
              "Found 5 at element %d in intArray\n", location );
```

```java
         else
            System.out.println( "5 not found in intArray" );

         // search intArray for the value 8763
         location = Arrays.binarySearch( intArray, 8763 );

         if ( location >= 0 )
            System.out.printf(
               "Found 8763 at element %d in intArray\n", location );
         else
            System.out.println( "8763 not found in intArray" );
      } // end main

      // output values in each array
      public static void displayArray( int[] array, String description )
      {
         System.out.printf( "\n%s: ", description );

         for ( int value : array )
            System.out.printf( "%d ", value );
      } // end method displayArray
   } // end class ArrayManipulations
```

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
filledIntArray: 7 7 7 7 7 7 7 7 7 7
intArray: 1 2 3 4 5 6
intArrayCopy: 1 2 3 4 5 6

intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray
```

Thank you.