## Declare arrays

## Initialize an array

## Use variable subscripts with an array

## Declare and use arrays of objects

## Search an array and use parallel arrays

## Pass arrays to and return arrays from methods

## **Array**

### A named list of data items

### All data items have the same type

## Declare an array variable

### The same way as declaring any simple variable

## **Subscript**

### An integer contained within square brackets

### Indicates one of the array's variables or elements

### A subscript that is too small or too large for an array is **out of bounds**

#### **An error message is generated**

#### An array's elements are numbered beginning with 0

### You can legally use any subscript from 0 through 19 when working with an array that has 20 elements

## When working with any individual array element, treat it no differently than a single variable of the same type

## A variable with a reference type, such as an array, holds a memory address where a value is stored

## **Array names**:

### Represent computer memory addresses

### Contain references

### Use plural and can include List

### salesList[], sales[], salesArray[]

## When you declare an array name:

### No computer memory address is assigned

### The array has the special value null

#### Unicode value '\u0000'

## Use the keyword new to define an array

### The array name acquires the actual memory address value

## int[] someNums = new int[10];

### Each element of someNums has a value of 0

## char array elements

### Assigned '\u0000'

## boolean array elements

### Automatically assigned the value false

## Strings and arrays of objects

### Assigned null by default

## Assign nondefault values to array elements upon creation

### int[] tenMults = {10, 20, 30, 40, 50, 60};

## An **initialization list** initializes an array

### Values are separated by commas and enclosed within curly braces

## **Populating an array**

### Providing values for all the elements in an array

# Using Variable Subscripts with an Array

## Scalar

### A primitive variable

## Power of arrays

### **Use subscripts that are variables rather than constant subscripts**

## When an application contains an array:

### Use every element of the array in some task

### Perform loops that vary the loop control variable

#### Start at 0

#### End at one less than the size of the array

## It is convenient to declare a symbolic constant equal to the size of the array

### final int NUMBER\_OF\_SCORES = 5;

## Field

### An instance variable

### Automatically assigned a value for every array created

## **length field:** number of elements in the array

### for(sub = 0; sub < **scoreArray.length**; ++sub)

#### scoreArray[sub] += 3;

## length is a **property** of the object

### Is a field

### Cannot be used as an array method

# Enhanced for (also called foreach)

## **Enhanced for loop**

### Allows you to cycle through an array without specifying starting and ending points for the loop control variable

### for(int val : scoreArray)

#### System.out.println(val);

# Using Part of an Array

## In cases when you do not want to use every value in an array

## Might be useful to you….

# Declaring and Using Arrays of Objects

## Create an array of Employee objects

### **Employee[] emp = new Employee[7];**

### Must call seven individual constructors

### final double PAYRATE = 6.35;

### for(int num = 0; num < NUM\_EMPLOYEES; ++num)

#### emp[num] = new Employee(101 + num, PAYRATE);

# Using the Enhanced for Loop with Objects

## **Use the enhanced for loop to cycle through an array of objects**

### Eliminates the need to use a limiting value

### Eliminates the need for a subscript following each element

### for(Employee worker : emp)

### System.out.println(worker.getEmpNum() + " " + worker.getSalary();

# Manipulating Arrays of Strings

## Create an array of Strings

### String[] deptNames = {"Accounting", "Human Resources", "Sales"};

### for(int dept = 0; a < deptNames.length; ++dept)

#### System.out.println(deptNames[dept]);

# Searching an Array and Using Parallel Arrays

## Determine whether a variable holds one of many valid values

### Use a series of if statements

#### Would get very large!

### Compare the variable to a series of valid values

#### Array in action

# Search the Array

## **Searching an array**

### Compare the variable to a list of values in an array

### Works but can waste resources, why?

### for(int item = 0; item < validValues.length; ++item)

### {

#### if(itemOrdered == validValues[item])

##### validItem = true;

### }

# Using Parallel Arrays

## **Parallel array**

### One with the same number of elements as another

### The values in corresponding elements are related

## An alternative for searching

### Use the while loop

## Searching an array for an exact match is not always practical

### discounts

## **Range match**

### Compare a value to the endpoints of numerical ranges

### Find the category in which a value belongs

### Look at the parallel arrays

## Pass a single array element to a method

### Same as passing a variable

## **Passed by value**

### A copy of the value is made and used in the receiving method

### All primitive types are passed this way

## **Reference types**

### The object holds a memory address where the values are stored

### **The receiving method gets a copy of the array's actual memory**

### The receiving method has the ability to alter the original values in the array elements

## A method can return an array reference

## Include square brackets with the return type in the method header

# Summary\_0

## Array

### A named list of data items

### All have the same type

## Array names

### Represent computer memory addresses

## Shorten many array-based tasks

### Use a variable as a subscript

## length field

### Contains the number of elements in an array

# Summary\_1

## You can declare arrays that hold elements of any type, including Strings and other objects

## Search an array to find a match to a value

## Perform a range match

## Pass a single array element to a method