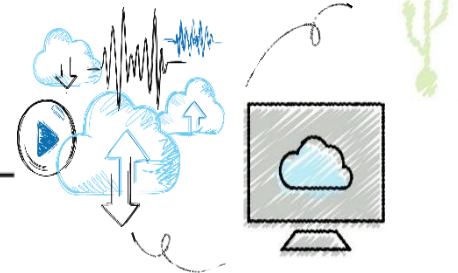# Java Programming, 9e

Chapter 8

Arrays

Objectives

- Declare an array

- 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 called **elements**
  - All data items have the same type

- Declare an array variable
  - The same way as declaring any simple variable
  - Insert a pair of square brackets after the type

```
double[] salesFigure;

int[] idNums;
```

CENGAGE

- Still need to reserve memory space

```
sale = new double[20];
double[] sale = new double[20];
```

- **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**
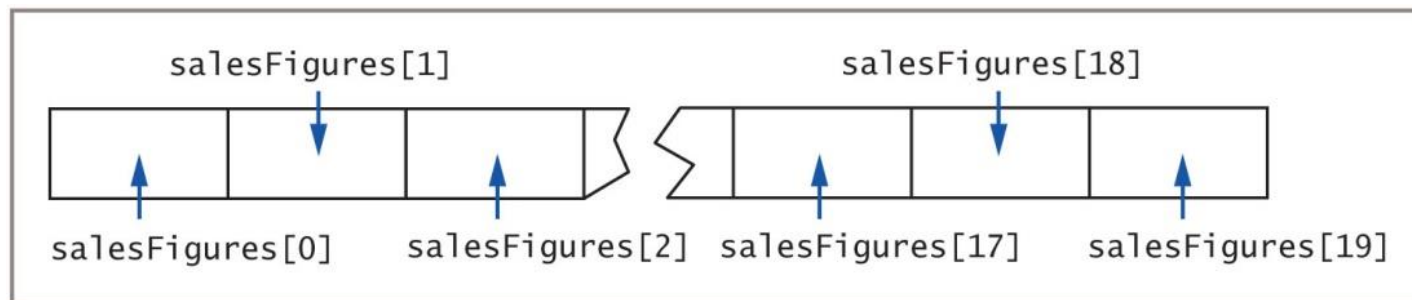   - An error message is generated

Declaring Arrays (3 of 4)

- 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
  - Example: `sale[0] = 2100.00;`

Declaring Arrays (4 of 4)

Figure 8-1 The first few and last few elements of an array of 20 `salesFigures` items in memory

Initializing an Array (1 of 3)

- 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

- When you declare an array name:
  - No computer memory address is assigned
  - The array has the special value `null`
    - Unicode value '\u0000'

CENGAGE

- 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`

- `String`s and arrays of objects
  - Assigned `null` by default

- Assign nondefault values to array elements upon creation

  ```
  int[] tenMult = {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

CENGAGE

- Power of arrays
  - Use subscripts that are variables rather than constant subscripts
  - Use a loop to perform array operations

```
for (sub = 0; sub < 5; ++sub)
    scoreArray[sub] += 3;
```

- 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;
```

CENGAGE

- 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` 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);
```

```java
import java.util.*;
public class AverageOfQuizzes
{
    public static void main(String[] args)
    {
        int[] scores = new int[10];
        int score = 0;
        int count = 0;
        int total = 0;
        final int QUIT = 999;
        final int MAX = 10;
        Scanner input = new Scanner(System.in);
        System.out.print("Enter quiz score or " +
            QUIT + " to quit      >>");
    }
}
```

Figure 8-4 The AverageOfQuizzes application (*continues*)

(*continued*)

```
        score = input.nextInt();
        while(score !=QUIT)                          Loop continues as long as user
        {                                            does not enter QUIT value
            score[count] = score;
            total += scores[count];
            ++count;
            if(count == MAX)
                score = QUIT;
            else
            {
                System.out.print("Enter next quiz score or " +
                    QUIT + " to quit >> ");
                score = input.nextInt();
            }
        }
        System.out.print("\nThe scores entered were: ")
        for(int x = 0; x < count; ++x)               The variable count
            System.out.print(scores[x] + " ");       is used to control
        if(count != 0)                               output.
            System.out.println("\n The average is " + (total * 1.0 / count));
        else
            System.out.println("No scores were entered.");
    }
}
```

Figure 8-4   The AverageOfQuizzes application

Declaring and Using Arrays of Objects

- Create an array of `Employee` objects

  ```
  Employee[] emp = new Employee[7];
  ```

  - Must call seven individual constructors

  ```
      final int START_NUM = 101;

      final double STARTING_SALARY = 15_000;

      for(int x = 0; x < emps.length; ++x)

        emps[x] = new Employee(START_NUM + x,
                     STARTING_SALARY);
  ```

CENGAGE

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 : emps)
      System.out.println(worker.getEmpNum() + " " +
      worker.getSalary();
```

CENGAGE

Manipulating Arrays of `Strings`

- Create an array of `Strings`

```
String[] deptNames = {"Accounting", "Human Resources",
"Sales"};
for(int a = 0; a < deptNames.length; ++a)
    System.out.println(deptNames[a]);
```

CENGAGE

- Determine whether a variable holds one of many valid values
  - Use a series of `if` statements
  - Compare the variable to a series of valid values

CENGAGE

- **Searching an array**
  - Compare the variable to a list of values in an array

```
for(int x = 0; x < validValues.length; ++x)
{
    if(itemOrdered == validValues[x])
        isValidItem = true;
}
```

CENGAGE

- **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

CENGAGE

```java
import javax.swing.*;
public class FindPrice
{
    public static void main(String[] args)
    {
        final int NUMBER_OF_ITEMS = 10;
        int[] validValues = {101, 108, 201, 213, 266,
            304, 311, 409, 411, 412};
        double[] prices = {0.29, 1.23, 3.50, 0.69, 6.79,
            3.19, 0.99, 0.89, 1.26, 8.00};
        String strItem;
        int itemOrdered;
        double itemPrice = 0.0;
        boolean isValidItem = false;
        strItem = JOptionPane.showInputDialog(null,
            "Enter the item number you want to order");
        itemOrdered = Integer.parseInt(strItem);
        for(int x = 0; x < NUMBER_OF_ITEMS; ++x)
        {
            if(itemOrdered == validValues[x])
            {
                isValidItem = true;
                itemPrice = prices[x];
            }
        }
        if(isValidItem)
            JOptionPane.showMessageDialog(null, The price for item " +
                itemOrdered + " is $" + itemPrice);
        else
            JOptionPane.showMessageDialog(null,
                "Sorry - invalid item entered");
    }
}
```

Corresponding price is pulled from `prices` array

Figure 8-9   The `FindPrice` application that accesses information in parallel arrays

```
for(int x = 0; x < NUMBER_OF_ITEMS; ++x)
{
    if(itemOrdered == validValues[x])
    {
        isValidItem = true;
        itemPrice = prices[x];
        x = NUMBER_OF_ITEMS
    }
}
```

Force the loop control variable to a value that stops the loop.

Figure 8-11   A for loop with an early exit

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

- **Range match**
  - Compare a value to the endpoints of numerical ranges
  - Find the category in which a value belongs

```java
import javax.swing.*;
public class FindDiscount
{
    public static void main(String[] args)
    {
        final int NUM_RANGES = 5;
        int[] discountRangeLimits = { 1, 13, 50, 100, 200};
        double[] discountRates =    {0.00, 0.10, 0.14, 0.18, 0.20};
        double customerDiscount;
        String strNumOrdered;
        int numOrdered;
        int sub = NUM_RANGES - 1;
        strNumOrdered = JOptionPane.showInputDialog(null,
            "How many items are ordered?");
        numOrdered = Integer.parseInt(strNumOrdered);
        while(sub >= 0 && numOrdered < discountRangeLimits[sub])
            --sub;
        customerDiscount = discountRates[sub];
        JOptionPane.showMessageDialog(null, "Discount rate for " +
            numOrdered + " items is " + customerDiscount);
    }
}
```

Figure 8-13   The FindDiscount class

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

```
public class PassArray
{
    public static void main(String)[] args)
    {
        final int NUM_ELEMENTS = 4;
        int[] someNums = {5, 10, 15, 20};
        int x;
        System.out.print("At start of main: ");
        for(x = 0; x < NUM_ELEMENTS; ++x)
            System.out.print(" " + someNums[x]);
        System.out.println();
        methodGetsArray(SomeNums);
        System.out.print("At end of main: ");
        for(x = 0; x < NUM_ELEMENTS; ++x)
            System.out.print(" " + someNums[x]);
        System.out.println();
    }
}
```

When an array is passed to a method, no brackets are used.

Figure 8-18   The PassArray class (*continues*)

(continued)

```java
public static void methodGetsArray(int[] arr)
{
    int x;
    System.out.print("At start of method arr holds: ");
    for(x = 0; x < arr.length; ++x)
        System.out.print(" " + arr[x]);
    System.out.println();
    for(x = 0; x < arr.length; ++x)
        arr[x] = 888;
    System.out.print(" and at end of method arr holds: ");
    for(x = 0; x < arr.length; ++x)
        System.out.print(" " + arr[x]);
    System.out.println();
}
}
```

Brackets are used in the parameter list to show that an array is passed.

Figure 8-18   The PassArray class

Returning an Array from a Method

- A method can return an array reference

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

Don't Do It

- Don't forget that the lowest array subscript is 0

- Don't forget that the highest array subscript is one less than the length

- Don't forget the semicolon following the closing curly brace in an array initialization list

- Don't forget that `length` is an array property and not a method

- Don't place a subscript after an object's field or method name when accessing an array of objects

- Don't assume that an array of characters is a string

- Don't forget that array names are references

- Don't use brackets with an array name when you pass it to a method

CENGAGE

- 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

- 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