

# **Chapter 9**

## **Introduction to Arrays**

Fundamentals of Java

---

# Objectives

- Write programs that handle collections of similar items.
- Declare array variables and instantiate array objects.
- Manipulate arrays with loops, including the enhanced `for` loop.

## Objectives (cont.)

- Write methods to manipulate arrays.
- Create parallel arrays and two-dimensional arrays.

# Vocabulary

- Array
- Element
- Enhanced `for` loop
- Index
- Initializer list
- Logical size

# Vocabulary (cont.)

- Multidimensional array
- One-dimensional array
- Parallel arrays
- Physical size
- Procedural decomposition

# Vocabulary (cont.)

- Ragged array
- Range-bound error
- Structure chart
- Subscript
- Two-dimensional array

# Conceptual Overview

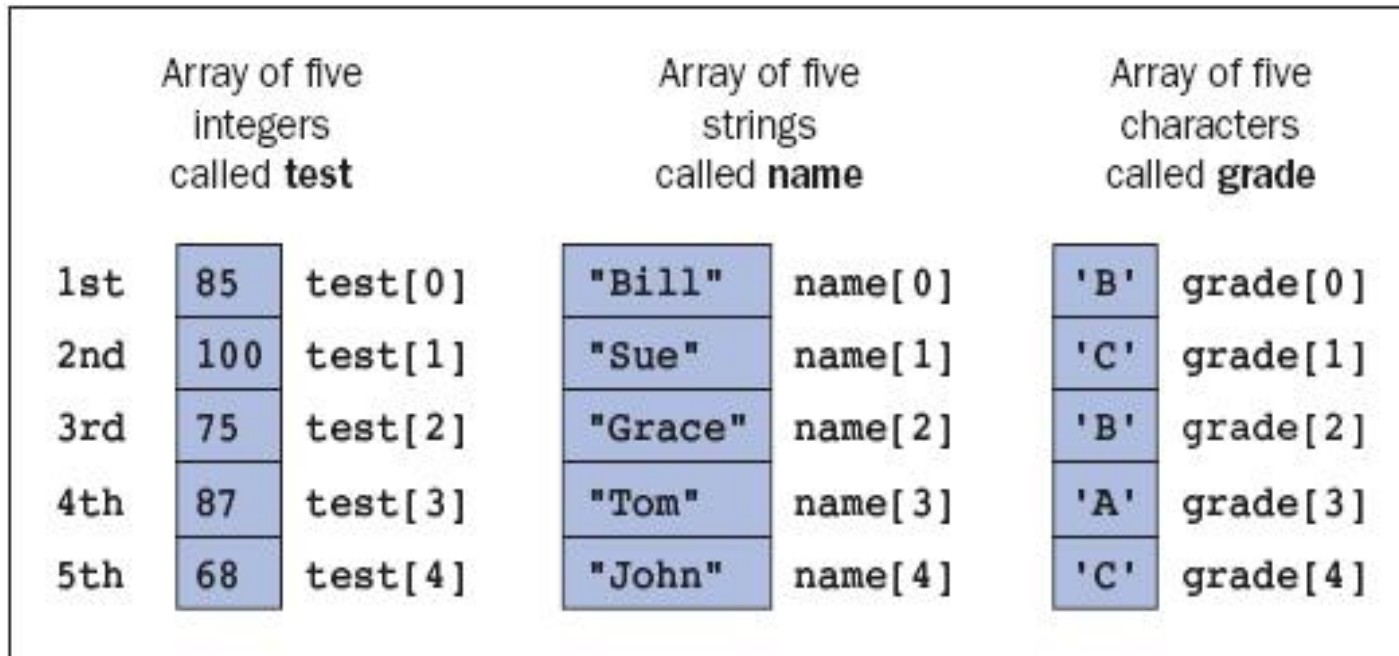


Figure 9-1: Three arrays, each containing five elements

# Conceptual Overview (cont.)

- **Elements:** The items in an array
- All array elements have the same data type.
  - Can be primitive or reference type
- Every array has a fixed **length** (size).
- Every array element has an **index** or **subscript**.
  - Appear within square brackets ( `[]` )

# Simple Array Manipulations

- Declaring an array of 500 integers:
  - `int[] abc = new int[500];`
- Syntax for referring to an array element:
  - `<array name>[<index>]`
  - `<index>` must be between 0 and the array's length minus 1
- **Subscript operator:** `[]`

# Simple Array Manipulations (cont.)

- Examples:

```
int[] abc = new int[500];  
  
int i = 3;  
int temp;  
double avFirstFive;  
  
abc[0] = 78;           // 1st element 78  
abc[1] = 66;           // 2nd element 66  
abc[2] = (abc[0] + abc[1]) / 2; // 3rd element average of first two  
abc[i] = 82;           // 4th element 82 because i is 3  
abc[i + 1] = 94;       // 5th element 94 because i + 1 is 4  
abc[499] = 76;        // 500th element 76
```

# Simple Array Manipulations (cont.)

- **Range-bound error:** Referring to a non-existent array index
  - `ArrayIndexOutOfBoundsException` thrown
    - `abc[-1] = 48;`
    - `abc[500] = 48;`
- **Interchanging adjacent array elements:**

```
abc[3] = 82;
abc[4] = 95;
i = 3;

temp = abc[i];           // temp        now equals 82
abc[i] = abc[i + 1];    // abc[i]      now equals 95
abc[i + 1] = temp;      // abc[i + 1] now equals 82
```

# Looping Through Arrays

- Using a loop to iterate through each element in an array is a common task.
  - Loop counter used as array index
  - Example applications:
    - Sum the elements of an array
    - Count occurrences of a value in an array
    - Search for a value in an array
    - Find first occurrence of a value in an array

# Looping Through Arrays (cont.)

- An array's `length` instance variable yields the size of the array.
- Example of loop structure for looping through an array of any size:

```
int sum;  
sum = 0;  
for (int i = 0; i < abc.length; i++)  
    sum += abc[i];
```

# Declaring Arrays

- Arrays are objects.
  - Must be instantiated before use
  - Declaration example:
    - `int[] abc;`
  - Instantiation example:
    - `abc = new int[5];`
  - Variations:
    - `int[] abc = new int[5];`
    - `int[] abc = new int[5], xyz = new int[10];`

# Declaring Arrays (cont.)

- Because arrays are objects, all rules that apply to objects apply to arrays.
  - Two array variables may refer to same array.
  - Arrays may be garbage collected.
  - Array variables may be set to `null`.
  - Arrays are passed by reference to methods.

# Declaring Arrays (cont.)

- Example of two array variables pointing to same array object:

```
int[] abc, xyz;  
abc = new int[5];  
  
xyz = abc;  
xyz[3] = 100;  
System.out.println (abc[3]);  
  
// Instantiate an array of five  
// integers  
// xyz and abc refer to the same array  
// Changing xyz changes abc as well.  
// 100 is displayed.
```

# Declaring Arrays (cont.)

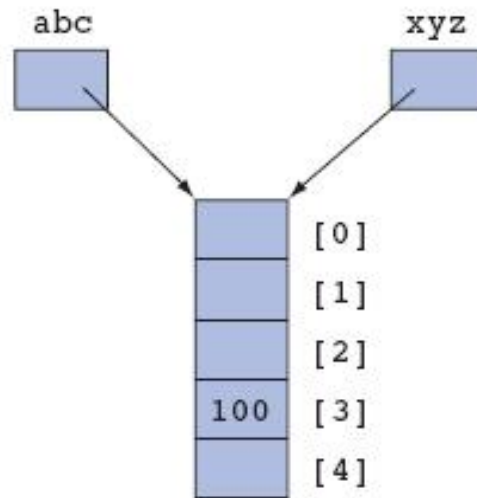


Figure 9-2: Two variables can refer to the same array object.

# Declaring Arrays (cont.)

- Arrays can be declared, instantiated, and initialized using an **initializer list**.

```
int[] abc = {1,2,3,4,5}; // abc now references an array of five integers.
```

- Arrays can contain collections of similar items.
  - Primitive or reference types
- Once set, the size of an array is fixed.

# Declaring Arrays (cont.)

- Example:

```
double[] ddd = new double[10];
char[] ccc = new char[10];
boolean[] bbb = new boolean[10];
String[] ggg = new String[10];
Student[] sss = new Student[10];
String str;

ddd[5] = 3.14;
ccc[5] = 'Z';
bbb[5] = true;
ggg[5] = "The cat sat on the mat.";
sss[5] = new Student();

sss[5].setName ("Bill");
str = sss[5].getName() + ggg[5].substring(7);
// str now equals "Bill sat on the mat."
```

# Working with Arrays That Are Not Full

- When an array is instantiated, it is filled with default values.
- An application might not fill all cells.
- **Physical size:** The maximum number of elements that an array can contain
- **Logical size:** The actual number of elements stored in an array

# Working with Arrays That Are Not Full (cont.)

- To work with arrays that are not full, the programmer must track the logical array size.
  - Declare an integer counter that will always indicate the number of elements.
  - Every time an element is added or removed, adjust the counter accordingly.
  - The counter indicates the logical size of the array and the next open position in the array.

# Working with Arrays That Are Not Full (cont.)

- Processing an array that is not full:

```
int[] abc = new int[50];
int size = 0;
```

... code that puts values into some initial portion of the array and sets the value of size ...

```
int sum = 0;
for (int i = 0; i < size; i++)
    sum += abc[i];
```

- Adding an element:

```
if (size < abc.length){
    abc[size] = anInt;
    size++;
}
```

# Parallel Arrays

- Two or more arrays of the same size that store complementary data
- Example: one array stores first names and a second stores corresponding ages.

```
String[] name = {"Bill", "Sue", "Shawn", "Mary", "Ann"};  
int[]     age  = {20,    21,    19,    24,    20};
```

# Two-Dimensional Arrays

- Every array discussed so far is a **one-dimensional** array.
  - Visualized as a *list* of items or values
- Arrays may have multiple dimensions.
- Two-dimensional arrays can be visualized as *tables* with rows and columns.
  - An “array of arrays”
    - Each element of a one-dimensional array contains another array.

# Two-Dimensional Arrays (cont.)

	col 0	col 1	col 2	col 3	col 4
row 0	00	01	02	03	04
row 1	10	11	12	13	14
row 2	20	21	22	23	24
row 3	30	31	32	33	34

Figure 9-3: Two-dimensional array with four rows and five columns

# Two-Dimensional Arrays (cont.)

- Declaring and instantiating a two-dimensional array example:

```
int[][] table;           // The variable table can reference a
                        // two-dimensional array of integers.
table = new int[4][5];  // Instantiate table as an array of size 4,
                        // each of whose elements will reference an array
                        // of 5 integers.
```

# Two-Dimensional Arrays (cont.)

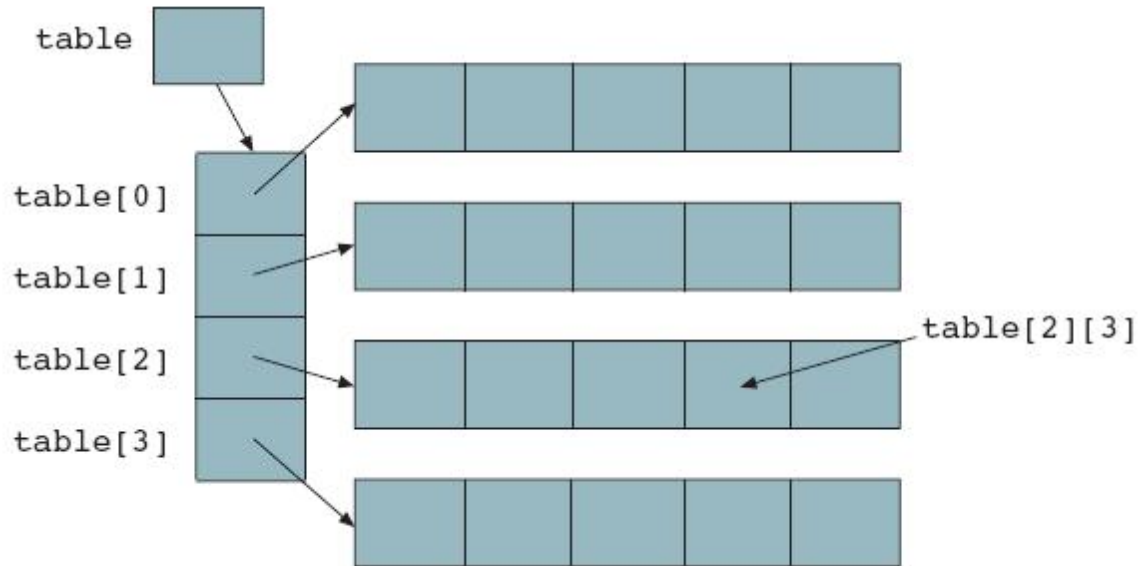


Figure 9-4: Another way of visualizing a two-dimensional array

# Two-Dimensional Arrays (cont.)

- Looping through a two-dimensional array:

```
int[] rowSum = new int[4];
for (int i = 0; i < table.length; i++){
    for (int j = 0; j < table[i].length; j++){
        rowSum[i] += table[i][j];
    }
}
```

# Two-Dimensional Arrays (cont.)

- Initializer lists for two-dimensional arrays:

```
int[][] table = {{ 0, 1, 2, 3, 4},      // row 0
                 {10,11,12,13,14},     // row 1
                 {20,21,22,23,24},     // row 2
                 {30,31,32,33,34}};    // row 3
```

- The rows in a two-dimensional array may have varying lengths.
  - **Ragged arrays**

# Using the Enhanced for Loop

- Provided in Java 5.0 to simplify loops in which each element in an array is accessed
  - From the first index to the last
  - Frees programmer from having to manage and use loop counters
- Syntax:

```
for (<temporary variable declaration> : <array object>)  
    <statement>
```

# Using the Enhanced for Loop (cont.)

```
// Example 9.3: Testing the enhanced for loop

public class TestForLoop{

    public static void main(String[] args){

        // Sum the elements in a one-dimensional array
        int[] abc = {2, 3, 4};
        int sum = 0;
        for (int element : abc)
            sum += element;
        System.out.println("First sum: " + sum);

        // Sum the elements in a two-dimensional array
        int[][] table = {{2, 3, 4}, {2, 3, 4}, {2, 3, 4}};
        sum = 0;
        for (int[] row : table)
            for (int element : row)
                sum += element;
        System.out.println("Second sum: " + sum);
    }
}
```

Example 9.3: Testing the enhanced for loop

# Using the Enhanced for Loop (cont.)

- Cannot be used to:
  - Move through an array in reverse, from the last position to the first position
  - Assign elements to positions in an array
  - Track the index position of the current element in an array
  - Access any element other than the current element on each pass

# Arrays and Methods

- Because arrays are objects, they are passed by reference to methods.
  - The method manipulates the array itself, not a copy.
    - Changes to array made in the method exist after method is complete.
- The method may create a new array and return it.

# Arrays and Methods (cont.)

- Method to search for a value in an array:

```
int search (int[] a, int searchValue){
    for (int i = 0; i < a.length; i++)
        if (a[i] == searchValue)
            return i;
    return -1;
}
```

- Method to sum the rows in a 2-D array:

```
int[] sumRows (int[][] a){
    int[] rowSum = new int[a.length];
    for (int i = 0; i < a.length; i++){
        for (int j = 0; j < a[i].length; j++){
            rowSum[i] += a[i][j];
        }
    }
    return rowSum;
}
```

# Arrays and Methods (cont.)

- Method to make a copy of an array and return it:

```
// First the method
int[] copyTwo (int[] original){
    int[] copy = new int[original.length];
    for (int i = 0; i < original.length; i++){
        copy[i] = original[i];
    }
    return copy;
}

// And here is how we call it.
int[] orig = {1,2,3,4,5};
int[] cp = copyTwo (orig);
```

# Arrays of Objects

- Arrays can hold references to objects of any type.
  - When instantiated, each cell has a value of `null`.
- Example:

```
// Declare and reserve 10 cells for student objects
Student[] studentArray = new Student[10];

// Fill array with students
for (int i = 0; i < studentArray.length; i++)
    studentArray[i] = new Student("Student " + i, 70+i, 80+i, 90+i);
```

# Case Study: Design Techniques

- **UML diagrams:** Industry standard for designing and representing a set of interacting classes
- **Structure charts:** May be used to depict relationships between classes and the order of methods called in a program
- **Procedural decomposition:** Technique of dividing a problem into sub-problems and correlating each with a method

# Case Study: Design Techniques (cont.)

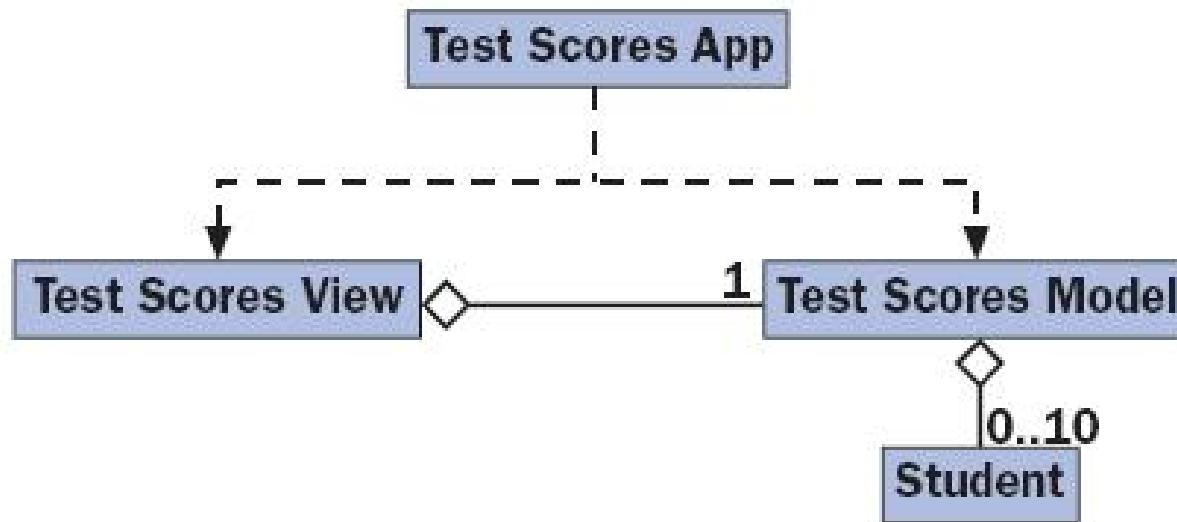


Figure 9-6: UML diagram of the classes in the student test scores program

# Case Study: Design Techniques (cont.)

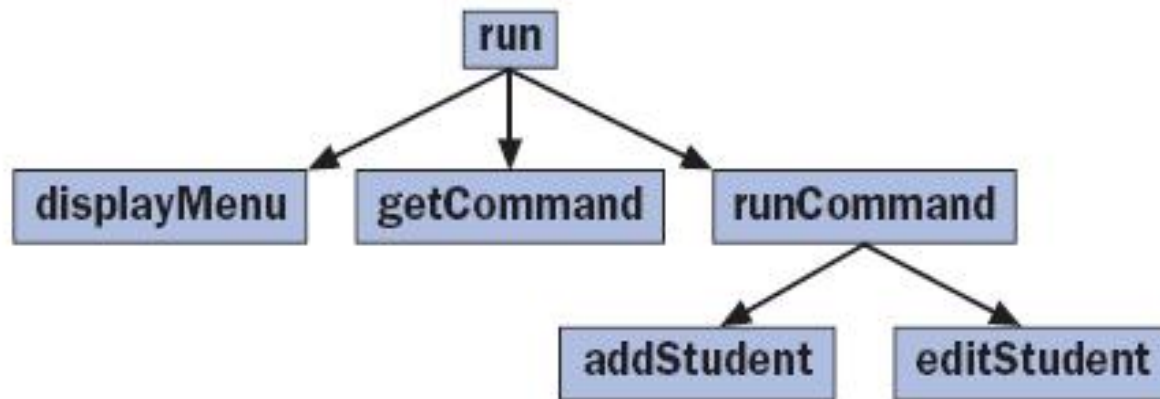


Figure 9-7: Structure chart for the methods  
of class `TestScoresView`

# Design, Testing, and Debugging Hints

- To create an array:
  - 1. Declare an array variable.
  - 2. Instantiate an array object and assign it to the array variable.
  - 3. Initialize the cells in the array with data, as appropriate.
- When creating a new array object, try to determine an accurate estimate of the number of cells required.

# Design, Testing, and Debugging Hints (cont.)

- Remember that array variables are `null` until they are assigned array objects.
- To avoid index out-of-bounds errors, remember that the index of an array cell ranges from 0 (the first position) to the length of the array minus 1.
- To access the last cell in an array, use the expression `<array>.length - 1`.

# Design, Testing, and Debugging Hints (cont.)

- Avoid having multiple array variables refer to the same array.
- To copy the contents of one array to another, do not use `A = B;` instead, write a copy method and use `A = arrayCopy(B);`.
- When an array is not full:
  - Track the current number of elements
  - Avoid index out-of-bounds errors

# Summary

- Arrays are collections of similar items or elements ordered by position.
- Arrays are useful when a program needs to manipulate many similar items, such as a group of students or a number of test scores.
- Arrays are objects.
  - Must be instantiated
  - Can be referred to by more than one variable

## Summary (cont.)

- An array can be passed to a method as a parameter and returned as a value.
- Parallel arrays are useful for organizing information with corresponding elements.
- Two-dimensional arrays store values in a row-and-column arrangement.

## Summary (cont.)

- An enhanced `for` loop is a simplified version of a loop for visiting each element of an array from the first position to the last position.