Name: _____

Java 1-Dimensional Arrays

What is an Array?

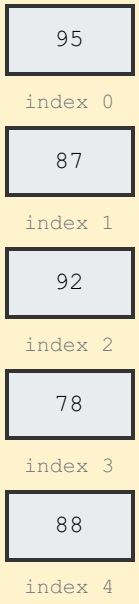
An **array** is a container that holds multiple values of the same data type in a single variable. Instead of creating many separate variables, you can store all related data in one organized structure. Arrays are like having numbered boxes in a row, where each box can hold one piece of data.

Simple Analogy: Arrays are Like Bookshelves

Think of an array like a bookshelf with numbered slots:

- **The entire bookshelf** = The array (holds multiple books)
- **Each book** = One element/value in the array
- ☐ **Shelf numbers (0, 1, 2, 3...)** = Index numbers (positions)
- **Fixed number of slots** = Array size (can't add more slots later)
- **All books are the same type** = All elements have same data type

Example: Bookshelf Array "scores"



Remember: We start counting from 0, not 1!

Key Array Concepts

- **Index:** The position number (starts at 0)
- **Element:** Each individual value stored in the array
- **Length:** How many elements the array can hold
- **Type:** All elements must be the same data type
- **Fixed Size:** Once created, size cannot change

How to Create Arrays

Method 1: Declare, Create, Then Assign

```
// Step 1: Declare the array
int[] scores;

// Step 2: Create the array with size
scores = new int[5];

// Step 3: Assign values
scores[0] = 95;
scores[1] = 87;
scores[2] = 92;
scores[3] = 78;
scores[4] = 88;
```

Method 2: Declare and Create Together

```
int[] scores = new int[5]; // Empty array with 5 slots

// Then assign values
scores[0] = 95;
scores[1] = 87;
// etc...
```

Method 3: Initialize with Values (Easiest!)

```
// Create and fill in one line
int[] scores = {95, 87, 92, 78, 88};
// Or with different data types:
String[] names = {"Alice", "Bob", "Charlie", "Diana"};
double[] prices = {19.99, 5.50, 12.75};
boolean[] flags = {true, false, true, true};
```

Accessing Array Elements

Reading and Writing Array Values

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

// Read values (get information from array)
int first = numbers[0]; // 10
int third = numbers[2]; // 30
int last = numbers[4]; // 50
System.out.println("First number: " + first);
System.out.println("Third number: " + third);
System.out.println("Last number: " + last);
// Write values (change information in array)
numbers[1] = 99; // Change 20 to 99
numbers[3] = numbers[3] + 10; // Add 10 to fourth element
```

```
System.out.println("Modified array:");
System.out.println(numbers[0] + ", " + numbers[1] + ", " +
numbers[2] + ", " + numbers[3] + ", " + numbers[4]);
```

Array Length Property

Finding Array Size

```
String[] colors = {"red", "green", "blue", "yellow"};

// Get the length (number of elements)
int size = colors.length; // 4 (no parentheses!)

System.out.println("Array has " + size + " elements");
System.out.println("First color: " + colors[0]);
System.out.println("Last color: " + colors[size - 1]); // Last index is always length-1

// Common pattern: loop through entire array
for (int i = 0; i < colors.length; i++) {
  System.out.println("Color " + i + ": " + colors[i]);
}
```

⚠ Common Array Mistakes

Mistake 1: Array Index Out of Bounds

```
int[] arr = {10, 20, 30}; // Length is 3, valid indexes: 0, 1, 2

// CRASH! Index 3 doesn't exist
int value = arr[3]; // ArrayIndexOutOfBoundsException

// CORRECT: Always check bounds
if (3 < arr.length) {
  int value = arr[3];
}
```

Mistake 2: Confusing Length with Last Index

```
int[] numbers = new int[5]; // Length = 5, last index = 4

// WRONG: This will crash
int last = numbers[numbers.length]; // Index 5 doesn't exist!

// CORRECT: Subtract 1 from length
int last = numbers[numbers.length - 1]; // Index 4
```

Looping Through Arrays

Method 1: Regular For Loop

```java
double[] grades = {85.5, 92.0, 78.5, 95.5, 88.0};

// Calculate average using regular for loop
double sum = 0;
for (int i = 0; i < grades.length; i++) {
  System.out.println("Grade " + (i + 1) + ": " + grades[i]);
  sum += grades[i];
}

double average = sum / grades.length;
System.out.println("Average grade: " + average);
```

Method 2: Enhanced For Loop (for-each)

```java
String[] fruits = {"apple", "banana", "orange", "grape"};

// Enhanced for loop - easier when you don't need the index
System.out.println("Fruits in our basket:");
for (String fruit : fruits) {
  System.out.println("- " + fruit);
}
// Read as: "for each fruit in fruits array, do this..."
// Note: You can't modify the array with enhanced for loop
```

Practical Array Examples

Example 1: Student Grade Manager

```java
import java.util.Scanner;

public class GradeManager {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    System.out.print("How many students? ");
    int numStudents = input.nextInt();
    // Create array to store grades
    double[] grades = new double[numStudents];

    // Input grades
    for (int i = 0; i < grades.length; i++) {
      System.out.print("Enter grade for student " + (i + 1) + ": ");
      grades[i] = input.nextDouble();
    }
    // Find highest and lowest grades
    double highest = grades[0];
    double lowest = grades[0];
    double sum = 0;

    for (double grade : grades) {
      if (grade > highest) {
        highest = grade;
      }
      if (grade < lowest) {
        lowest = grade;
```

```
      }
      sum += grade;
    }

    double average = sum / grades.length;

    // Display results
    System.out.println("\n=== Grade Report ===");
    System.out.println("Number of students: " + numStudents");
    System.out.println("Highest grade: " + highest);
    System.out.println("Lowest grade: " + lowest");
    System.out.println("Average grade: " + average);

    input.close();
  }
}
```

Example 2: Search and Count

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

// Count how many times target appears
int count = 0;
for (int num : numbers) {
  if (num == target) {
    count++;
  }
}
System.out.println("Number " + target + " appears " + count + " times");
// Find first position of target
int firstPosition = -1; // -1 means "not found"
for (int i = 0; i < numbers.length; i++) {
  if (numbers[i] == target) {
    firstPosition = i;
    break; // Stop searching once found
  }
}

if (firstPosition != -1) {
  System.out.println("First " + target + " found at index " + firstPosition);
} else {
  System.out.println(target + " not found in array");
}
```

Example 3: Array Sorting (Simple Bubble Sort)

```
int[] scores = {64, 89, 55, 92, 78, 45, 91};

System.out.println("Original scores:");
for (int score : scores) {
  System.out.print(score + " ");
}
System.out.println();
// Simple bubble sort (smallest to largest)
for (int i = 0; i < scores.length - 1; i++) {
  for (int j = 0; j < scores.length - i - 1; j++) {
```

```
    if (scores[j] > scores[j + 1]) {
      // Swap elements
      int temp = scores[j];
      scores[j] = scores[j + 1];
      scores[j + 1] = temp;
    }
  }
}
System.out.println("Sorted scores:");
for (int score : scores) {
  System.out.print(score + " ");
}
System.out.println();
```

Common Array Operations

Operation	Code Example	What It Does
Create empty array	int[] arr = new int[5];	Creates array with 5 slots, all initialized to 0
Create with values	int[] arr = {1, 2, 3, 4, 5};	Creates array and fills it with given values
Get array length	int size = arr.length;	Returns number of elements in array
Access element	int value = arr[2];	Gets value at index 2
Modify element	arr[2] = 99;	Changes value at index 2 to 99
Find last element	int last = arr[arr.length - 1];	Gets the last element in array

Example 4: Array Copy and Modification

```
int[] original = {1, 2, 3, 4, 5};
int[] copy = new int[original.length];

// Manual copy (element by element)
for (int i = 0; i < original.length; i++) {
  copy[i] = original[i];
}

// Modify the copy
for (int i = 0; i < copy.length; i++) {
  copy[i] *= 2; // Double each value
}

// Display both arrays
System.out.print("Original: ");
for (int num : original) {
  System.out.print(num + " ");
}
System.out.println();

System.out.print("Copy (doubled): ");
for (int num : copy) {
  System.out.print(num + " ");
}
System.out.println();
```

Array Best Practices

- **Always check bounds** before accessing elements
- **Use meaningful names** for arrays (scores, not arr)
- **Use .length** instead of hardcoding size
- **Initialize with values** when you know them
- **Use enhanced for loops** when you don't need index
- **Remember arrays start at index 0**

Common Array Mistakes

- **Using wrong index** (like arr[arr.length])
- **Confusing length with last index**
- **Not checking for empty arrays**
- **Trying to change array size** after creation
- **Using = to compare arrays** (compares references)
- **Forgetting arrays are objects** (passed by reference)

Example 5: Real-World Application - Store Inventory

```
import java.util.Scanner;
public class StoreInventory {
  public static void main(String[] args) {
    // Store product information
    String[] products = {"Apples", "Bananas", "Oranges", "Grapes", "Strawberries"};
    double[] prices = {2.99, 1.89, 3.49, 4.99, 5.99};
    int[] quantities = {50, 30, 25, 20, 15};
    Scanner input = new Scanner(System.in);
    // Display inventory
    System.out.println("=== Store Inventory ===");
    System.out.println("Product Price Quantity Total Value");
    System.out.println("-----------------------------------------------");

    double totalInventoryValue = 0;
    for (int i = 0; i < products.length; i++) {
      double itemValue = prices[i] * quantities[i];
      totalInventoryValue += itemValue;
      System.out.println(products[i] + " " + prices[i] + " " + quantities[i] + " " +
itemValue);
    }
    System.out.println("-----------------------------------------------");
    System.out.println("Total Inventory Value: $" + totalInventory + "%.2f");
    // Find most expensive item
    int mostExpensiveIndex = 0;
    for (int i = 1; i < prices.length; i++) {
      if (prices[i] > prices[mostExpensiveIndex]) {
        mostExpensiveIndex = i;
      }
    }
    System.out.println("Most expensive item: " + products[mostExpensiveIndex] +
" at $" + prices[mostExpensiveIndex]);
    // Check for low stock (less than 25 items)
    System.out.println("\nLow Stock Alert:");
    boolean foundLowStock = false;
    for (int i = 0; i < quantities.length; i++) {
      if (quantities[i] < 25) {
        System.out.println("- " + products[i] + ": " + quantities[i] + " left");
        foundLowStock = true;
```

```
        }
      }

      if (!foundLowStock) {
        System.out.println("All items have sufficient stock.");
      }
      input.close();
    }
}
```

Array Memory Concept

Understanding how arrays work in memory:

- **Array variable** stores a reference (address) to the actual array in memory
- **When you pass arrays to methods,** you're passing the reference, not a copy
- **Multiple variables can reference the same array**
- **Arrays are objects** - they have methods and properties like .length

```
int[] arr1 = {1, 2, 3};
int[] arr2 = arr1; // arr2 points to same array as arr1

arr2[0] = 99; // This changes arr1[0] too!

System.out.println(arr1[0]); // Prints 99, not 1
```

When to Use Arrays

Use Arrays When:

- **You have multiple related values** of the same type
- **You know the approximate size** in advance
- **You need fast access** by position/index
- **You want to process data in loops**
- **Order matters** for your data

Examples:

- Student test scores
- Daily temperatures
- Shopping cart items
- Game high scores

Consider Alternatives When:

- **Size changes frequently** (use ArrayList)
- **You need to insert/delete often** (use ArrayList)
- **You have mixed data types** (use objects)
- **You need key-value pairs** (use HashMap)
- **Size is completely unknown** (use ArrayList)

Note: Arrays are faster than ArrayList but less flexible!

Your Turn: Write Your Own Definition

What is a 1-dimensional array in Java? How would you explain it to a friend?

Write your definition in your own words:

Fill in the code to complete these array operations:

```
// Create an array of 5 integers
int[] numbers = _____________________;

// Set the first element to 10
_____________________ = 10;

// Get the last element of the array
int lastElement = _____________________;

// Print the length of the array
System.out.println("Array length: " + _____________________);
```

Describe three real-world situations where you would use an array:

1. _____

2. _____

3. _____

Explain why array indexes start at 0 instead of 1:

What would happen if you try to access arr[arr.length] in an array? Why?