Single-Dimensional Arrays

teacher: Майер С.Ф.
sfmayer@sfedu.ru

The Problem — Why Arrays?

- Imagine you need to store 100 test scores for a class.
- Without arrays, you would need 100 separate variables:

```
int score0 = 85;
int score1 = 92;
int score2 = 78;
// ... 97 more lines!
```

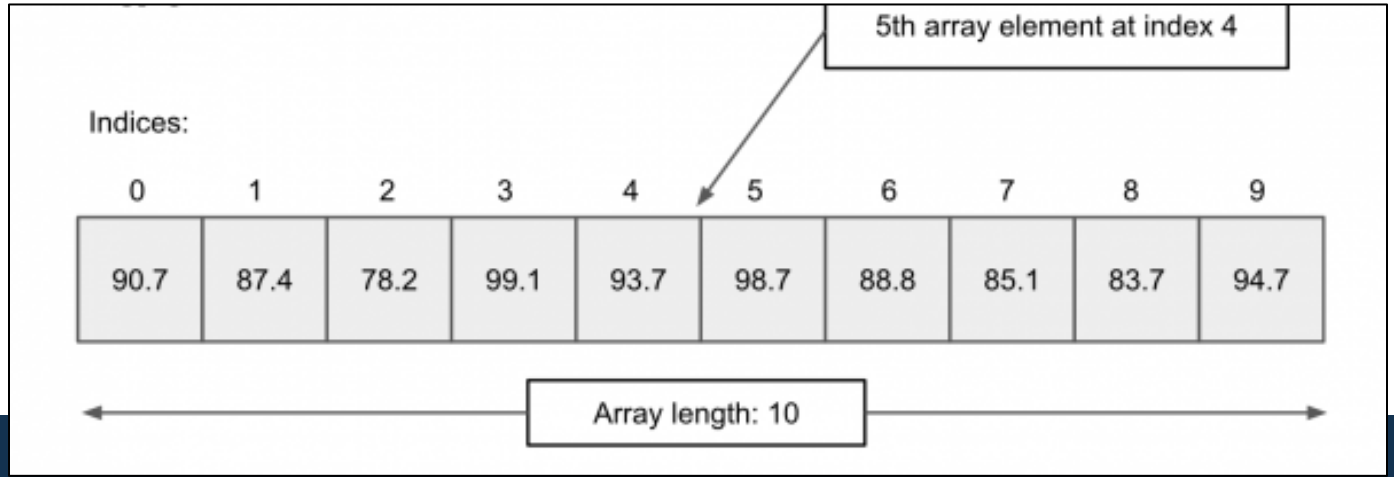
This is:

- **Tedious** to declare
- **Impossible** to loop through
- **Hard** to maintain

Solution: Use an **array** — a single variable that can hold multiple values.

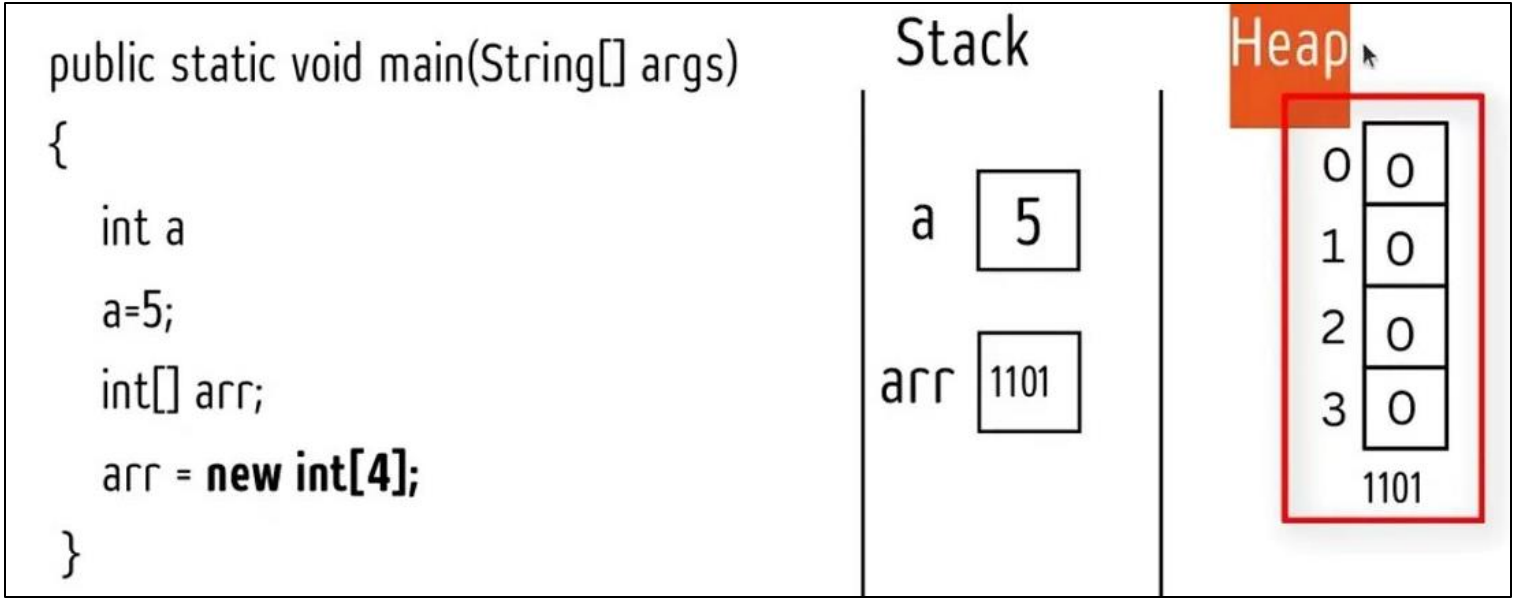
Key Array Characteristics

- An **array** is a data structure that stores a fixed-size sequential collection of elements of the **same type**.
- Instead of declaring many individual variables, you declare one array variable and use **indexed variables**.
  - **Zero-Indexed**: The first element is at index 0, the last at index **length - 1**.
  - **Fixed Size**: Once created, the size cannot change.
  - **Homogeneous**: All elements must be of the same data type.



Key Array Characteristics

- **Reference Type**: An array variable holds a reference (memory address) to the actual array object.



Declaring Array Variables

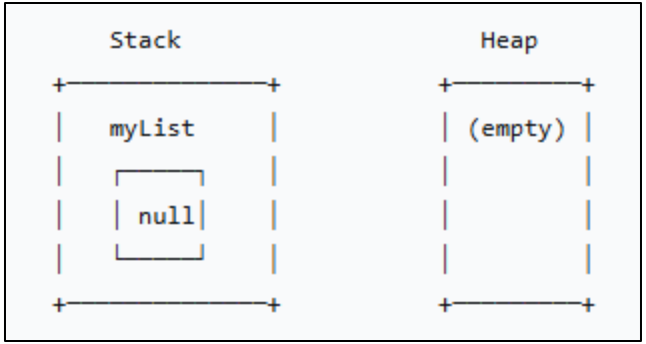
To use an array, you must declare a variable that can reference it.

Syntax:

```
elementType[] arrayRefVar;
// or
elementType arrayRefVar[];  // Allowed, but less preferred
```

Examples:

```
double[] myList;     // Declares an array of double
int[] scores;        // Declares an array of int
String[] names;      // Declares an array of String
```



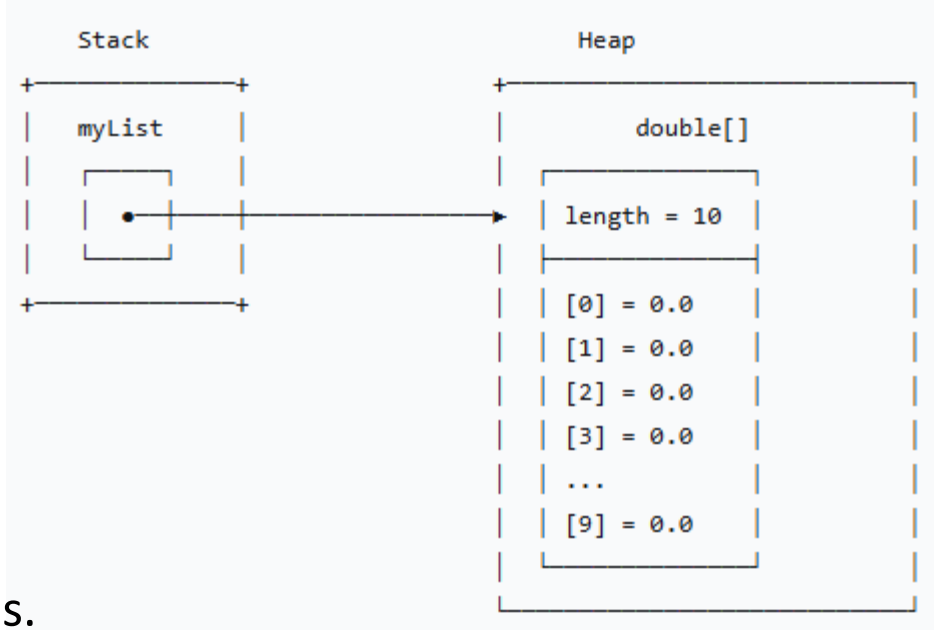
Important: Declaring an array variable does not create the array. It only allocates space for a reference. The value is null initially.

Creating Arrays (Allocating Memory)

Use the new keyword to create an array and allocate memory.

Syntax:

```
arrayRefVar = new elementType[arraySize];
```



Example:

```
double[] myList;
myList = new double[10];
```

After this:

- Memory is allocated for 10 double values.
- Each element gets a **default value** (0.0 for double).
- myList now contains a **reference** to this array.

Combining Declaration and Creation

You can declare and create an array in one statement:

Examples:

```
double[] myList = new double[10];
int[] scores = new int[25];
String[] names = new String[50];
```

Data Type	Default Value
byte, short, int, long	0
float, double	0.0
char	'\u0000' (null character)
boolean	false
Objects (String, etc.)	null

Array Size and Default Values

Getting the size: Use the length property.

```
double[] myList = new double[10];
int size = myList.length; // Returns 10
```

Important: Once created, the size is fixed. You cannot add or remove elements.

Accessing Array Elements (Indices)

Access elements using the **index** inside square brackets.

Syntax:

```
arrayRefVar[index] = value; // Assign a value
value = arrayRefVar[index]; // Read a value
```

Example:

```
double[] myList = new double[10];
// Assign values
myList[0] = 5.6;
myList[1] = 4.5;
myList[2] = 3.3;
// Use indexed variables like regular variables
myList[2] = myList[0] + myList[1];  // myList[2] becomes 10.1
// Loop to initialize all elements
for (int i = 0; i < myList.length; i++) {
    myList[i] = i;
}
```

Output (after loop):

```
myList[0] = 0
myList[1] = 1
myList[2] = 2
...
```

Array Initializers (Shorthand Notation)

Java provides a shorthand for declaring, creating, and initializing an array in one statement.

Syntax:

elementType[] arrayRefVar = {value0, value1, ..., valuek};

Example:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This is equivalent to:

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

Important: Cannot use this shorthand after declaration:

```
double[] myList;
myList = {1.9, 2.9, 3.4, 3.5};  // INVALID!
```

Common Array Processing Operations

- Arrays are typically processed using **loops** because:

1. All elements are the same type → can be processed uniformly.
2. The size is known → perfect for counted loops (for loop).

- Common operations :
  - Initializing with user input
  - Initializing with random values
  - Displaying array elements
  - Summing all elements
  - Finding the largest element
  - Finding the index of the largest element
  - Shuffling elements
  - Shifting elements

Initializing Arrays (User Input & Random)

1. Initializing with User Input:

```
import java.util.Scanner;

double[] myList = new double[10];
Scanner input = new Scanner(System.in);

System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++) {
    myList[i] = input.nextDouble();
}
```

2. Initializing with Random Values (0.0 to 100.0):

```
double[] myList = new double[10];

for (int i = 0; i < myList.length; i++) {
    myList[i] = Math.random() * 100;
}
```

Displaying Arrays

For numeric arrays, you must loop through each element:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};

for (int i = 0; i < myList.length; i++) {
    System.out.print(myList[i] + " ");
}
```

Output:

1.9 2.9 3.4 3.5

Note: `System.out.println(myList)` for numeric arrays prints the reference, not the elements!

For char **arrays,** you can print directly:

```
char[] city = {'M', 'o', 's', 'c', 'o', 'w'};
System.out.println(city);  // Output: Moscow
```

Algorithms

Example

Summing All Elements

To do: Use a variable `total` to accumulate the sum of array elements: 1.9, 2.9, 3.4, 3.5.

Output:

Sum: 11.7

Example

Summing All Elements

To do: Use a variable `total` to accumulate the sum of array elements: 1.9, 2.9, 3.4, 3.5.

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
double total = 0;

for (int i = 0; i < myList.length; i++) {
    total += myList[i];
}

System.out.println("Sum: " + total);
```

Output:

Sum: 11.7

Example

Finding the Largest Element

To do: Use a variable max to track the largest value of array elements: 1.9, 2.9, 3.4, 3.5.

Output:

Maximum: 3.5

Example

Finding the Largest Element

To do: Use a variable max to track the largest value of array elements: 1.9, 2.9, 3.4, 3.5.

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
double max = myList[0];  // Assume first is largest

for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) {
        max = myList[i];
    }
}

System.out.println("Maximum: " + max);
```

Output:

Maximum: 3.5

Example

Finding the Index of the Largest Element

Sometimes you need the **position** of the largest element, not just the value. Output it. (Array (`1, 5, 3, 4, 5, 5`))

Output:

Max value: 5.0

First occurrence at index: 1

Example

Finding the Index of the Largest Element

Sometimes you need the **position** of the largest element, not just the value. Output it.

```
double[] myList = {1, 5, 3, 4, 5, 5};
double max = myList[0];
int indexOfMax = 0;

for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) {
        max = myList[i];
        indexOfMax = i;
    }
}

System.out.println("Max value: " + max);
System.out.println("First occurrence at index: " + indexOfMax);
```

Output:

Max value: 5.0

First occurrence at index: 1

Example

Shuffling Elements

To do: Randomly reorder the elements in an array.

Output (example):

3 6 1 4 2 5

Example

Shuffling Elements

To do: Randomly reorder the elements in an array.

```
double[] myList = {1, 2, 3, 4, 5, 6};

for (int i = 0; i < myList.length; i++) {
    // Generate random index j between 0 and length-1
    int j = (int)(Math.random() * myList.length);
    // Swap myList[i] with myList[j]
    double temp = myList[i];
    myList[i] = myList[j];
    myList[j] = temp;
}

// Print shuffled array
for (double value : myList) {
    System.out.print(value + " ");
}
```

Output (example):

3 6 1 4 2 5

Example

Shifting Elements

To do: Shift all elements one position **to the left**.

Output:

2 3 4 5 1

To shift right: Start from the end and move backwards.

Example

Shifting Elements

To do: Shift all elements one position to the left.

```
double[] myList = {1, 2, 3, 4, 5};
double temp = myList[0];  // Save the first element

// Shift elements left
for (int i = 1; i < myList.length; i++) {
    myList[i - 1] = myList[i];
}

// Move the first element to the end
myList[myList.length - 1] = temp;

// Print result
for (double value : myList) {
    System.out.print(value + " ");
}
```

Output:

2 3 4 5 1

To shift right: Start from the end and move backwards.

Simplifying Coding — Month Names Example

Arrays can dramatically simplify code that would otherwise require long if-else or switch statements.

Without array (tedious):

```
if (monthNumber == 1) System.out.println("January");
else if (monthNumber == 2) System.out.println("February");
// ... 10 more cases!
```

Example Run:
Enter month number (1-12): 5
Month: May

With array (elegant):

```
String[] months = {
    "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"
};

System.out.print("Enter month number (1-12): ");
int monthNumber = input.nextInt();
System.out.println("Month: " + months[monthNumber - 1]);
```

foreach Loop

Foreach Loop

Java provides a simplified loop for traversing arrays sequentially.

Syntax:

```
for (elementType element : arrayRefVar) {
  // Process element
}
```

Example:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};

// Display all elements
for (double e : myList) {
    System.out.println(e);
}
```

Output:

1.9
2.9
3.4
3.5

Important: The foreach loop:

- Cannot modify array elements (you get a copy).
- Cannot traverse in reverse order or skip elements.
- Use indexed for loop when you need to modify or access indices.

Common Mistakes

Array Index Out of Bounds & Off-by-One Errors

ArrayIndexOutOfBoundsException: Occurs when you access an index outside 0 to length-1.

```
int[] arr = new int[5];
arr[5] = 10;  // ERROR: index 5 is out of bounds (max is 4)
```

Off-by-one: Using <= instead of <

```java
// INCORRECT – causes exception when i == length
for (int i = 0; i <= arr.length; i++) {
    System.out.println(arr[i]);
}

// CORRECT
for (int i = 0; i < arr.length; i++) {
    System.out.println(arr[i]);
}
```

Starting index at 1 instead of 0:

```
// INCORRECT – skips first element
for (int i = 1; i < arr.length; i++) { ... }
```

Prevention: Use length property and foreach loop when possible.

Passing Arrays to Methods

Passing Arrays to Methods

You can pass arrays to methods just like primitive values.

Example method:

```
public static void printArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
    System.out.println();
}
```

Output:

3 1 2 6 4 2

Invoking the method:

```
int[] numbers = {3, 1, 2, 6, 4, 2};
printArray(numbers);  // Pass the array

// Or use an anonymous array
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Anonymous array: Created without a variable name.

Pass-by-Value vs. Array Reference

Important Distinction:

Primitive types: The **value** is passed. Changes inside the method do NOT affect the original.

Array types: The **reference** is passed. Changes inside the method DO affect the original array.

Why?

The array variable stores a **reference** (memory address).

When passed, a copy of this reference is passed.

Both the original and the copy point to the **same array object**.

Visual:

```
Primitive:   int x = 5;  →  method(int n)  →  n is a COPY
Array:       int[] arr = {1,2}; → method(int[] a) → a refers to SAME array
```

Demonstrating Pass-by-Value with Primitives vs. Arrays

```
public class TestPassArray {
    public static void main(String[] args) {
        int x = 1;
        int[] y = {2, 3};

        System.out.println("Before: x=" + x + ", y[0]=" + y[0]);
        m(x, y);
        System.out.println("After: x=" + x + ", y[0]=" + y[0]);
    }

    public static void m(int number, int[] array) {
        number = 1001;
        array[0] = 5555;
    }
}
```

Output:
Before: x=1, y[0]=2
After: x=1, y[0]=5555

Explanation:

- x (primitive) → unchanged
- y[0] (array element) → changed because array references the same array

Returning an Array from a Method

A method can return an array. The return type is `elementType[]`.

Example: Reverse an array

```
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
// Usage
public static void main(String[] args) {
    int[] list1 = {1, 2, 3, 4, 5, 6};
    int[] list2 = reverse(list1);

    System.out.print("Reversed: ");
    for (int value : list2) {
        System.out.print(value + " ");
    }
}
```

Output:

Reversed: 6 5 4 3 2 1

Summary

1. **Arrays** store multiple values of the same type in a single variable, accessed by index.
2. **Declaration:** `elementType[] arrayRefVar;`
3. **Creation:** `arrayRefVar = new elementType[size];`
4. **Initialization:** Use array initializers `{value0, value1, ...}` or loops.
5. **Index:** Starts at `0`, ends at `length - 1`.
6. **Processing:** Use `for` loops or foreach loops for iteration.
7. **Passing to methods:** Arrays are passed by reference — changes inside methods affect the original array.
8. **Returning arrays:** Methods can return arrays using `return new elementType[]{...};`
9. **Common pitfalls:** Off-by-one errors, index out of bounds, confusing primitives with array references.