

# Programming 1

## Week 08 – One-Dimensional Array

# Introduction to Arrays

- Primitive variables are designed to hold only one value at a time.
- An **array** is an indexed collection of data elements of the same type.
- **Indexed** means that the array elements are numbered (starting at 0).
- The restriction of the **same type** is an important one, because arrays are stored in consecutive memory cells. Every cell must be the same type (and therefore, the same size).
- An array can store any type of data but **only one type of data at a time**.

The diagram illustrates an array structure. On the left, the text 'Name of array (c)' has an arrow pointing to the first element's index 'c[ 0 ]'. Below this, the text 'Index (or subscript) of the element in array c' has an arrow pointing to the index '11' in the last element 'c[ 11 ]'. The array itself is represented as a vertical column of 12 yellow rectangular cells, each containing a numerical value. The indices 'c[ 0 ]' through 'c[ 11 ]' are listed to the left of each cell.

c[ 0 ]	-45
c[ 1 ]	6
c[ 2 ]	0
c[ 3 ]	72
c[ 4 ]	1543
c[ 5 ]	-89
c[ 6 ]	0
c[ 7 ]	62
c[ 8 ]	-3
c[ 9 ]	1
c[ 10 ]	6453
c[ 11 ]	78

# Creating Arrays (1 of 3)

- An array is an object, so it needs an object reference.

```
// Declare a reference to an array that will hold integers.  
int[] numbers;
```

- The next step creates the array and assigns its address to the `numbers` variable.

```
// Create a new array that will hold 6 integers.  
numbers = new int[6];
```

0	0	0	0	0	0
index 0	index 1	index 2	index 3	index 4	index 5

## Creating Arrays (2 of 3)

- It is possible to declare an array reference and create it in the same statement.

```
int[] numbers = new int[6];
```

- Arrays may be of any type.

```
float[] temperatures = new float[100];
```

```
char[] letters = new char[41];
```

```
long[] units = new long[50];
```

```
double[] sizes = new double[1200];
```

## Creating Arrays (3 of 3)

- The array size must be a non-negative number.
- It may be a literal value, a constant, or variable.

```
final int ARRAY_SIZE = 6;  
int[] numbers = new int[ARRAY_SIZE];
```

- Once created, an array size is fixed and cannot be changed.

# Accessing the Elements of an Array

20	0	0	0	0	0
numbers[0]	numbers[1]	numbers[2]	numbers[3]	numbers[4]	numbers[5]

- An array is accessed by:
  - the reference name
  - a subscript that identifies which element in the array to access.

**numbers[0] = 20; //pronounced "numbers sub zero"**

# Inputting and Outputting Array Elements

- Array elements can be treated as any other variable.
- They are simply accessed by the same name and a subscript.
- Array subscripts can be accessed using variables (such as for loop counters).

# Bounds Checking

- Array indexes always start at zero and continue to (array length - 1).

```
int values = new int[10];
```

- This array would have indexes 0 through 9.
- In `for` loops, it is typical to use *i*, *j*, and *k* as counting variables.
  - It might help to think of *i* as representing the word **index**.



# Array Length (1 of 2)

- Arrays are objects and provide a public field named `length` that is a constant that can be tested.

```
double[] temperatures = new double[25];
```

- The length of this array is 25.
- The length of an array can be obtained via its `length` constant.

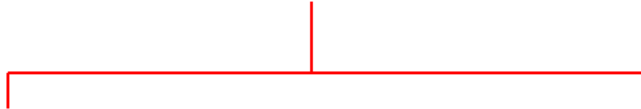
```
int size = temperatures.length;
```

- The variable `size` will contain 25.

## Array Length (2 of 2)

- For example, the following loop steps through the `values` array, and displays the contents of each element:

The loop repeats as long as  
`index` is less than the length of  
the `values` array



```
for (int index = 0; index < values.length; index++)  
    System.out.println(values[index]);
```

# Off-by-One Errors

- It is very easy to be off-by-one when accessing arrays.

```
// This code has an off-by-one error.  
int[] numbers = new int[100];  
for (int i = 1; i <= 100; i++)  
    numbers[i] = 99;
```

- Here, the equal sign allows the loop to continue on to index 100, where 99 is the last index in the array.
- This code would throw an `ArrayIndexOutOfBoundsException`.

# Array Initialization

- When relatively few items need to be initialized, an initialization list can be used to initialize the array.

```
int[] days = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

- The numbers in the list are stored in the array in order:
  - `days[0]` is assigned 31,
  - `days[1]` is assigned 28,
  - `days[2]` is assigned 31,
  - `days[3]` is assigned 30,
  - etc.

# Alternate Array Declaration

- Previously we showed arrays being declared:

```
int[] numbers;
```

- However, the brackets can also go here:

```
int numbers[];
```

- These are equivalent but the first style is typical.

- Multiple arrays can be declared on the same line.

```
int[] numbers, codes, scores;
```

- With the alternate notation each variable must have brackets.

```
int numbers[], codes[], scores;
```

- The `scores` variable in this instance is simply an `int` variable.

# Processing Array Contents (1 of 2)

- Processing data in an array is the same as any other variable.

```
grossPay = hours[3] * payRate;
```

- Pre and post increment works the same:

```
int[] score = {7, 8, 9, 10, 11};  
score[4]++;
```

# Processing Array Contents (2 of 2)

- Array elements can be used in relational operations:

```
if (cost[20] < cost[0])  
{  
    //statements  
}
```

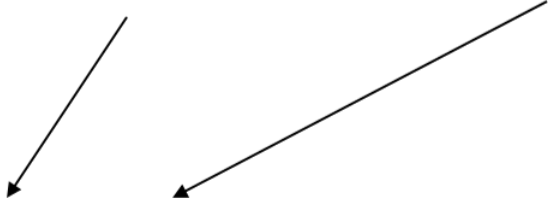
- They can be used as loop conditions:

```
while (value[count] != 0)  
{  
    //statements  
}
```

# Array Size (1 of 2)

- The `length` constant can be used in a loop to provide automatic bounding.

Index subscripts start at 0 and end at one **less than** the array length.



```
for(int i = 0; i < temperatures.length; i++)
{
    System.out.println("Temperature " + i ": "
                        + temperatures[i]);
}
```



## Array Size (2 of 2)

- You can let the user specify the size of an array:

```
int numTests;  
int[] tests;  
Scanner keyboard = new Scanner(System.in);  
System.out.print("How many tests do you have? ");  
numTests = keyboard.nextInt();  
tests = new int[numTests];
```

# Useful Array Operations (1 of 3)

- Finding the Highest Value

```
int [] numbers = new int[50];  
int highest = numbers[0];  
for (int i = 1; i < numbers.length; i++)  
{  
    if (numbers[i] > highest)  
        highest = numbers[i];  
}
```

# Useful Array Operations (2 of 3)

- Finding the Lowest Value

```
int lowest = numbers[0];
for (int i = 1; i < numbers.length; i++)
{
    if (numbers[i] < lowest)
        lowest = numbers[i];
}
```

# Useful Array Operations (3 of 3)

- Summing Array Elements:

```
int total = 0; // Initialize accumulator
for (int i = 0; i < units.length; i++)
    total += units[i];
```

- Averaging Array Elements:

```
double total = 0; // Initialize accumulator
double average;   // Will hold the average
for (int i = 0; i < scores.length; i++)
    total += scores[i];
average = total / scores.length;
```

# The Enhanced `for` Loop (1 of 2)

- Simplified array processing (read only)
- Always goes through all elements
- In a regular for loop, we have to create and control an index to visit elements. However, we only care about the values of the elements in the array most of the time, not the value of the index.
- General format:

```
for (datatype elementVariable : array)  
    statement;
```

# The Enhanced `for` Loop (2 of 2)

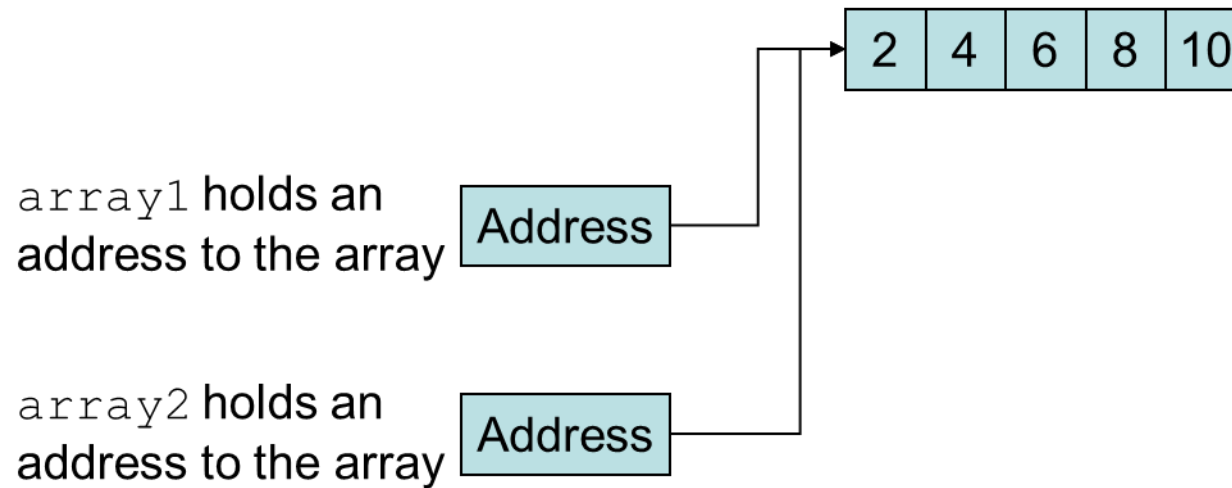
## Example:

```
int[] numbers = {3, 6, 9};  
for(int val : numbers)  
{  
    System.out.println("The next value is " + val);  
}
```

# Copying Arrays (1 of 2)

- This is **not** the way to copy an array.

```
int[] array1 = { 2, 4, 6, 8, 10 };  
int[] array2 = array1; // This does not copy array1.
```



## Copying Arrays (2 of 2)

- You cannot copy an array by merely assigning one reference variable to another.
- You need to copy the individual elements of one array to another.

```
int[] firstArray = {5, 10, 15, 20, 25 };  
int[] secondArray = new int[5];  
for (int i = 0; i < firstArray.length; i++)  
    secondArray[i] = firstArray[i];
```

- This code copies each element of `firstArray` to the corresponding element of `secondArray`.



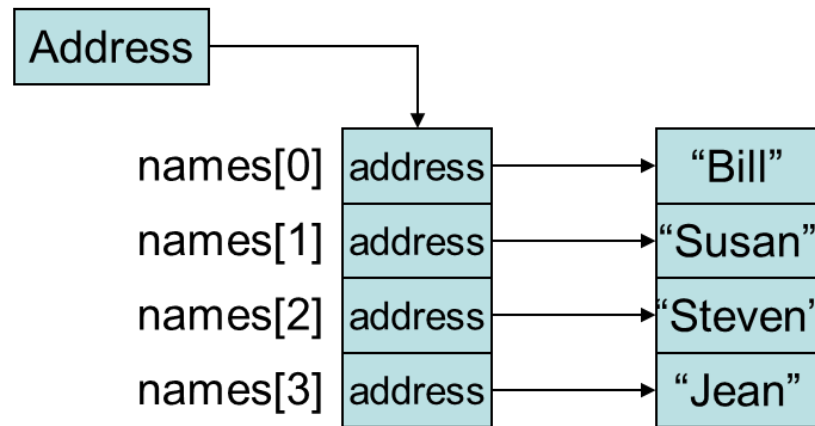
# String Arrays

- Arrays are not limited to primitive data.
- An array of `String` objects can be created:

```
String[] names = { "Bill", "Susan", "Steven", "Jean" };
```

The `names` variable holds the address to the array.

A `String` array is an array of references to `String` objects.



# Calling `String` Methods On Array Elements

- `String` objects have several methods, including:
  - `toUpperCase`
  - `equals`
  - `charAt`
  - Etc.,
- Each element of a `String` array is a `String` object.
- Methods can be used by using the array name and index as before.

```
System.out.println(names[0].toUpperCase());  
char letter = names[3].charAt(0);
```

# Demo: Compute sum and average of an array of test scores

```
double sum = 0, avg;
double [] scores = new double[10];
    // read the scores and store them in the array
for(int i = 0; i<scores.length; i++){
    System.out.print("Please enter score # " + (i+1) + ": ");
    scores[i] = input.nextDouble();
}
for(int i = 0; i < scores.length; i++)
    sum += scores[i]; // sum = sum + scores[i]

avg = sum/scores.length;
System.out.printf("The average of the scores is: %.2f %n", avg);
```