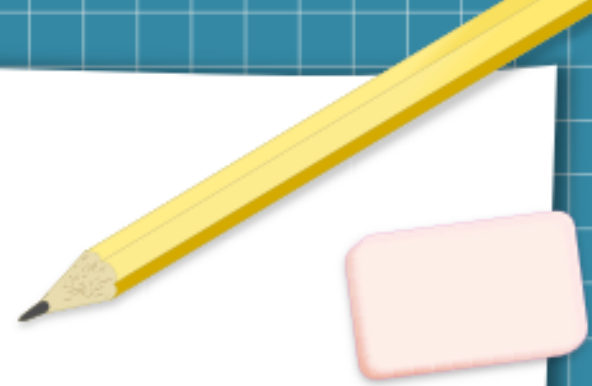


# Fair Use Notice



**The material used in this presentation i.e., pictures/graphs/text, etc. is solely intended for educational/teaching purpose, offered free of cost to the students for use under special circumstances of Online Education due to COVID-19 Lockdown situation and may include copyrighted material - the use of which may not have been specifically authorised by Copyright Owners. Its application constitutes Fair Use of any such copyrighted material as provided in globally accepted law of many countries. The contents of presentations are intended only for the attendees of the class being conducted by the presenter.**



# Arrays

Data Structure and Algorithms  
Practical # 03

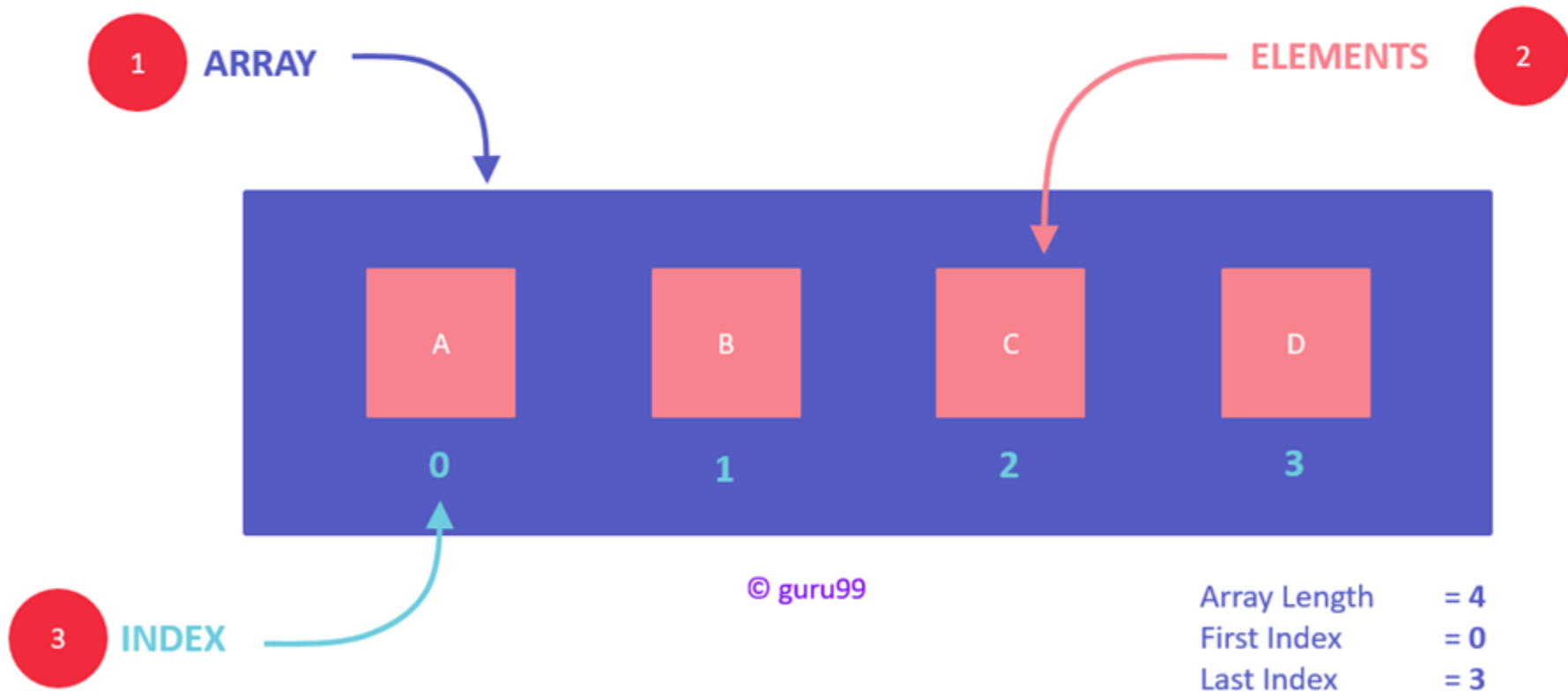
# Array Data Structure



- An **array** is an indexed sequence of components
  - Typically, the array occupies sequential storage locations
  - The length of the array is determined when the array is created, and cannot be changed
  - Each component of the array has a fixed, unique index
  - Indices range from a **lower bound** to an **upper bound**
- Any component of the array can be inspected or updated by using its index
  - This is an efficient operation:  $O(1)$  = constant time

# Concept Diagram

## CONCEPT DIAGRAM

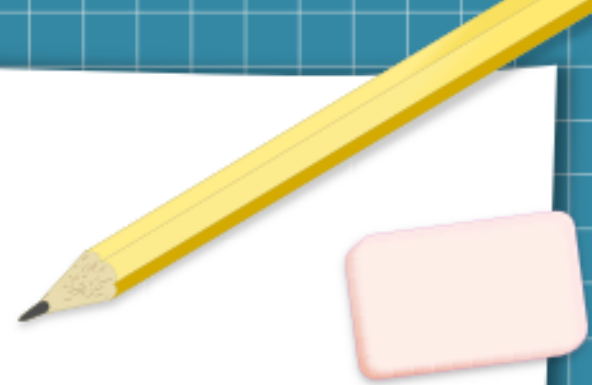


# Abstract Data Types

A yellow pencil with a pink eraser is positioned in the top right corner of the slide, pointing towards the title.

- Abstract data types, commonly abbreviated ADTs, are a way of classifying data structures based on how they are used and the behaviors they provide.
- They do not specify how the data structure must be implemented but simply provide a minimal expected interface and set of behaviors.

# Array as an ADT



- An array is an **Abstract Data Type**
  - The array type has a set of *values*
  - The values are all the possible arrays
- The array type has a set of *operations* that can be applied uniformly to each of these values
  - The only operation is *indexing*
  - Alternatively, this can be viewed as two operations:
    - *inspecting* an indexed location
    - *storing into* an indexed location
- It's *abstract* because the implementation is hidden: all access is via the defined operations

# Other Operations in Arrays



- **Traversal:** Processing each element in the list
- **Searching:** Finding the location of the element with a given value or the record with a given key.
- **Insertion:** Adding a new element
- **Deletion:** Removing an element
- **Sorting:** Arranging the elements in some type of order.



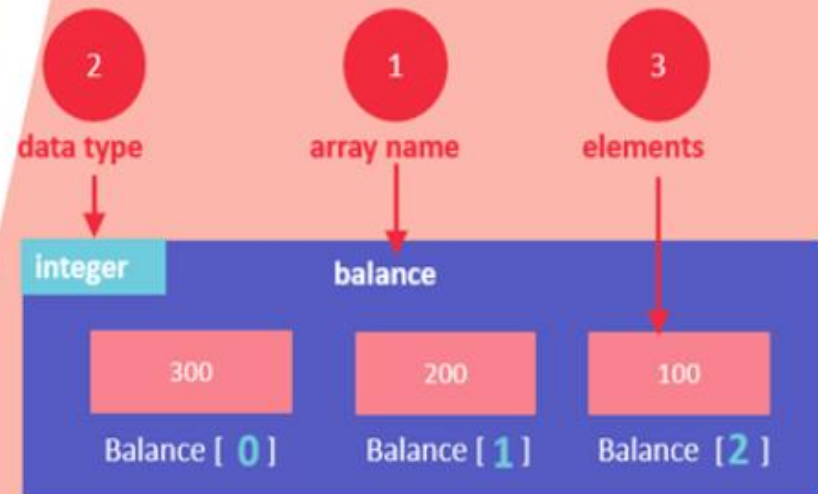
## UNDERSTAND SYNTAX

```
int[] balance = new int[]{300,200,100};
```

```
int[] balance = {300,200,100};
```

```
int[] balance = new int[3];  
balance[0] = 300;  
balance[1] = 200;  
balance[2] = 100;
```

(1) Array name, (2) data type and (3) elements are the fundamental parts of an array





# Traversing an Array

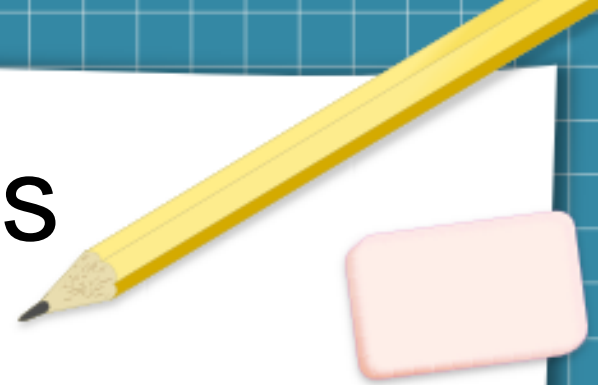


```
int[] balance = {300,200,100};  
  
for(int i : balance){  
    System.out.println(i);  
}
```

OR

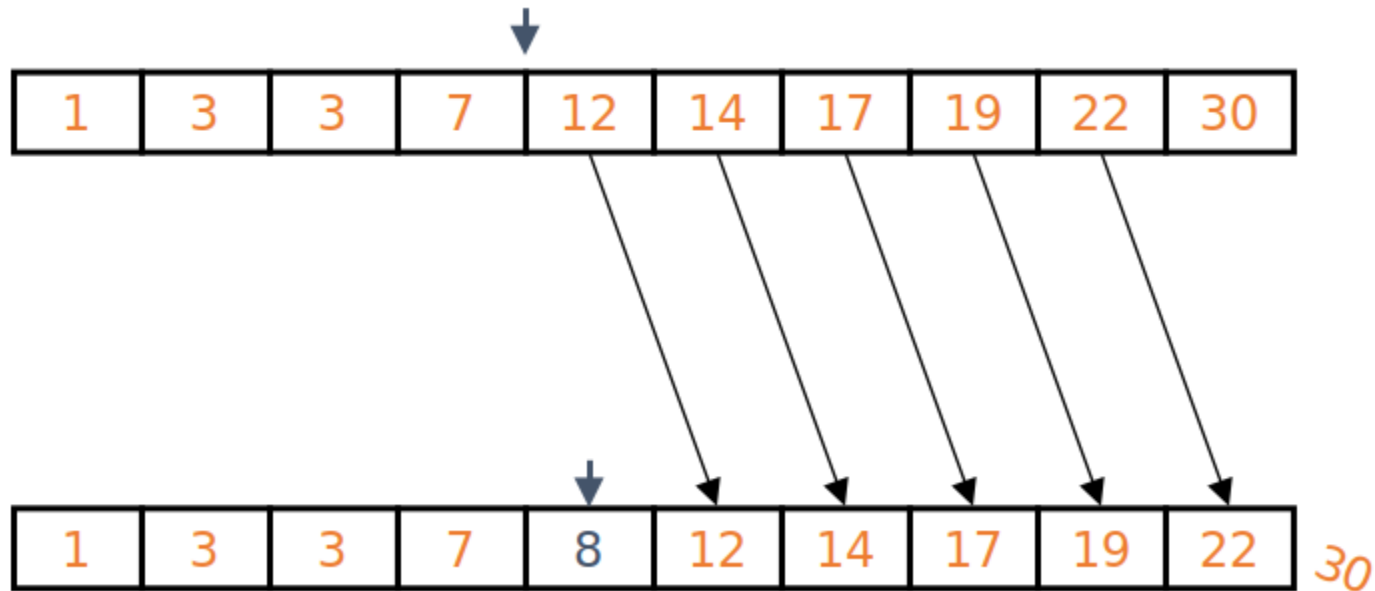
```
int[] balance = {300,200,100};  
  
for(int i=0; i<balance.length; i++){  
    System.out.println(i);  
}
```

# Swapping Elements



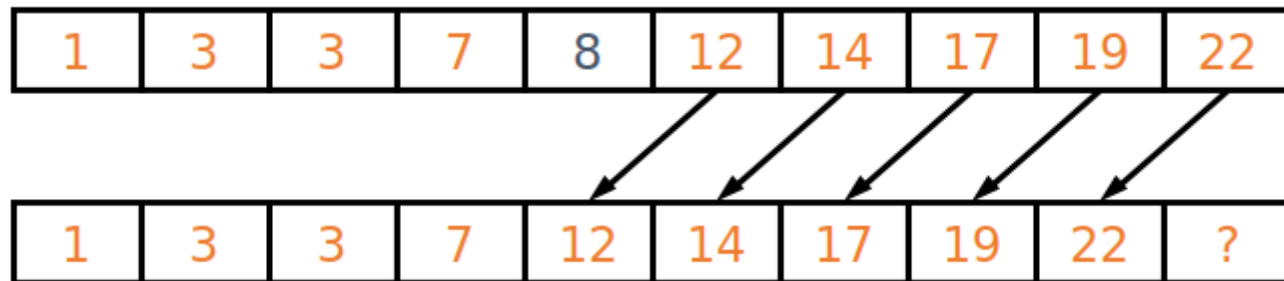
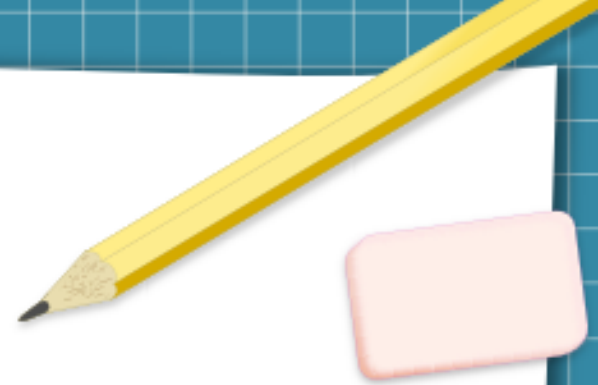
```
public void swap(int[] a, int i, int j)
{
    int ai= a[i];
    int aj =a[j];
    if (ai==aj) return;
    a[i]=aj;
    a[j]=ai;
}
```

# Insertion in Array



```
public void insert(int[] a, int index, int value)
{
    for(int i=a.length-1; i>index; i--)
    {
        a[i] = a[i-1];
    }
    a[index]=value;
}
```

# Deletion in Arrays



# Array Methods provided by Java

static int	<code>binarySearch(int[] a, int fromIndex, int toIndex, int key)</code> Searches a range of the specified array of ints for the specified value using the binary search algorithm.
static int	<code>binarySearch(long[] a, int fromIndex, int toIndex, long key)</code> Searches a range of the specified array of longs for the specified value using the binary search algorithm.
static int[]	<code>copyOf(int[] original, int newLength)</code> Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length.
static int[]	<code>copyOfRange(int[] original, int from, int to)</code> Copies the specified range of the specified array into a new array.
static boolean	<code>equals(int[] a, int[] a2)</code> Returns true if the two specified arrays of ints are <i>equal</i> to one another.
static void	<code>sort(int[] a)</code> Sorts the specified array into ascending numerical order.
static String	<code>toString(int[] a)</code> Returns a string representation of the contents of the specified array.


```
import java.util.Arrays;
class ArrayMethods
{
    public static void printArray(int[] array){
        for(int i:array)
            System.out.println(i);
    }

    public static void main(String args[])
    {
        int[] arr = {4,5,3,8,4,7};

        System.out.println("Printing original array...");
        printArray(arr);

        Arrays.sort(arr);
        System.out.println("Printing sorted array...");
        printArray(arr);
    }
}
```





```
int[] arrCopy = Arrays.copyOf(arr,6);  
System.out.println("Printing copied array...");  
printArray(arrCopy);
```

```
int[] arrCopyRange = Arrays.copyOfRange(arr,2,10);  
System.out.println("Printing copied range of array...");  
printArray(arrCopyRange);
```

```
System.out.println(Arrays.equals(arr,arrCopy));
```

```
}
```

```
}
```

# Multi-Dimensional Arrays



- Multidimensional arrays are arrays of arrays with each element of the array holding the reference of other array.
- A multidimensional array is created by appending one set of square brackets ([]) per dimension.

```
int[][] intArray = new int[10][20]; //a 2D array or matrix  
int[][][] intArray = new int[10][20][10]; //a 3D array
```

# Two Dimensional Array

- Two dimensional arrays allows us to store data that are recorded in table. For example:
- Table contains 12 items, we can think of this as a matrix consisting of 4 rows and 3 columns.

		<i>columns</i>				
		<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>
<i>rows</i>	<i>0</i>	<i>0,0</i>	<i>0,1</i>	<i>0,2</i>	<i>0,3</i>	<i>0,4</i>
	<i>1</i>	<i>1,0</i>	<i>1,1</i>	<i>1,2</i>	<i>1,3</i>	<i>1,4</i>
	<i>2</i>	<i>2,0</i>	<i>2,1</i>	<i>2,2</i>	<i>2,3</i>	<i>2,4</i>
	<i>3</i>	<i>3,0</i>	<i>3,1</i>	<i>3,2</i>	<i>3,3</i>	<i>3,4</i>

	Item1	Item2	Item3	Item4
Salesgirl #1	10	15	30	10
Salesgirl #2	14	30	33	50
Salesgirl #3	200	32	1	4

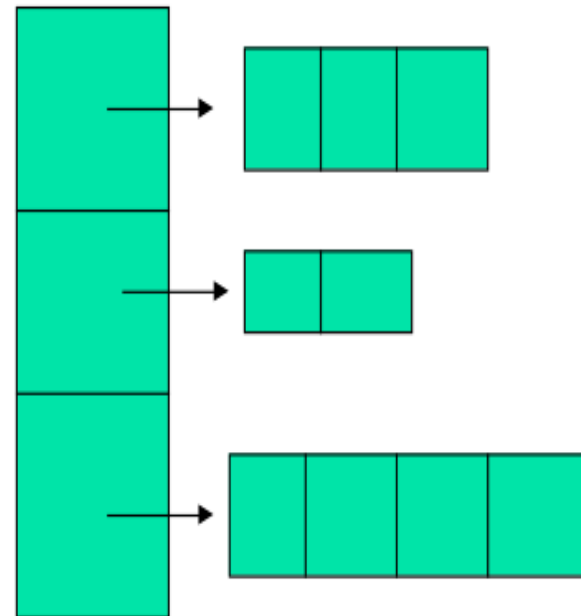
# 2-D Array Manipulations



- Declaration:
  - `int myArray [][];`
- Creation:
  - `myArray = new int[4][3]; // OR`
  - `int myArray [][] = new int[4][3];`
- Initialization:
  - Single Value;
    - `myArray[0][0] = 10;`
  - Multiple values:
    - `int tableA[2][3] = {{10, 15, 30}, {14, 30, 33}};`
    - `int tableA[][] = {{10, 15, 30}, {14, 30, 33}};`

# Variable Size 2-D Arrays

- Java treats multidimensional arrays as “arrays of arrays”. It is possible to declare a 2D arrays as follows:
  - `int a[][] = new int [3][];`
  - `a[0]= new int [3];`
  - `a[1]= new int [2];`
  - `a[2]= new int [4];`



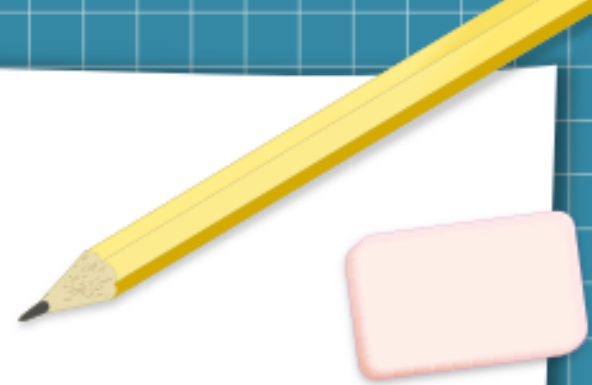
# Traversing a 2-D Array



```
class MultiArray{  
    public static void main(String args[]){  
        //Two dimensional array:  
        int[][] twoD_arr = new int[10][20];  
  
        //Three dimensional array:  
        int[][][] threeD_arr = new int[10][20][30];  
  
        int[][] arr = {{1, 2, 3}, {4, 5}};  
  
        for (int i = 0; i < arr.length; i++) {  
            for (int j = 0; j < arr[i].length; j++){  
                System.out.print(arr[i][j]+" ");  
            }  
  
            System.out.println();  
        }  
    }  
}
```



# Lab Tasks



1. Write a Java Code a Array of length 100 and fill it with Random int Values for testing purpose
2. Write a Java program to check if two arrays are equal.
3. Use all of the array method discussed above in your java code but array should not be of type integer.
4. Write a method in java with float as its return type that takes array as input and return average as output.

5. Write a method in java program to find the

# Lab Rubrics



Rubrics	Proficient	Adequate	Poor
<b>Programming Algorithms</b>	Produce correct results with efficient algorithms <b>(0.4)</b>	Produce results with inefficient algorithms <b>(0.2)</b>	Unable to design algorithms <b>(0.0)</b>
<b>Originality</b>	Submitted work shows large amount of original thought <b>(0.3)</b>	Submitted work shows some amount of original thought <b>(0.2)</b>	Submitted work shows no original thought <b>(0.0)</b>
<b>Troubleshooting</b>	Easily traced and corrected faults <b>(0.3)</b>	Traced and corrected faults with some guidance <b>(0.2)</b>	No troubleshooting skills <b>(0.0)</b>

Thank You

