**Unit-10(Holding Collection of Data)**

**Array**in array is a collection of similar type of elements which has contiguous memory location.Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.



### **Advantages**

* **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
* **Random access:** We can get any data located at an index position.

### **Disadvantages**

* **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, **collection framework** is used in Java which grows automatically.

### **Types of Array in java**

There are three types of array.

1. Single Dimensional Array
2. Multidimensional Array
3. Jagged Array

**Syntax:**  
dataType[] arr;

**Instantiation of an Array**   
arr=new datatype[size];

**Example:**

public class App {

    public static void main(String[] args) throws Exception {

        int[] a=new int[5];//declaration and instantiation

        a[0]=10;//initialization

        a[1]=20;

        a[2]=70;

        a[3]=40;

        a[4]=50;

        //traversing array

        for(int i=0;i<a.length;i++)//length is the property of array

        System.out.println(a[i]);

    }

}

## **Declaration, Instantiation and Initialization of Java Array**

public class App {

    public static void main(String[] args) throws Exception {

        int a[]={33,3,4,5};//declaration, instantiation and initialization

        //printing array

        for(int i=0;i<a.length;i++)//length is the property of array

        System.out.println(a[i]);

    }

}

## **For-each Loop for Java Array**

public class App {

    public static void main(String[] args) throws Exception {

        int arr[]={33,3,4,5};

        //printing array using for-each loop

        for(int i:arr)

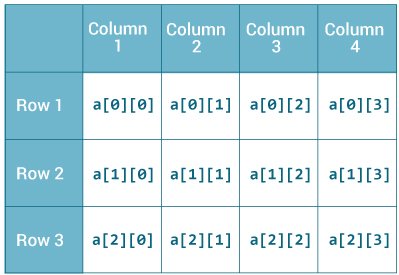
        System.out.println(i);

    }

}

## **Multidimensional Array in Java**

In such case, data is stored in row and column based index (also known as matrix form).



# Syntax: dataType[][] arrayRefVar;

**Instantiate Multidimensional Array** int[][] arr=new int[3][3];//3 row and 3 column

# Example of Multidimensional Java Array

public class App {

    public static void main(String[] args) throws Exception {

              //declaring and initializing 2D array

              int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

              //printing 2D array

              for(int i=0;i<3;i++)

              {

              for(int j=0;j<3;j++)

                {

                System.out.print(arr[i][j]+" ");

                }

              System.out.println();

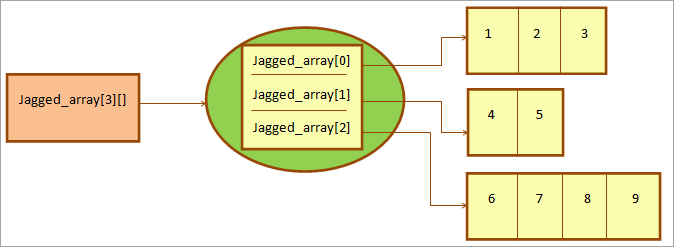
            }

}

}

## **Jagged Array in Java**

If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.



public class App {

    public static void main(String[] args) throws Exception {

int[][] matrix = new int[][]{{ 1, 2, 3 },{ 4, 5, 6, 7 },{ 8, 9 }};

        for (int i = 0; i < matrix.length; i++) //row

        {

            for (int j = 0; j < matrix[i].length; j++) //column

            {

                System.out.print(matrix[i][j]+" ");

            }

            System.out.println();

        }

}

}

Output:

1 2 3

4 5 6 7

8 9

Lab: Write a Java program to initialize and display jagged array elements with sum of each row.

public class App {

    public static void main(String[] args) throws Exception {

        int[][] matrix = new int[][]{{1, 2, 3}, {4, 5, 6, 7}, {8, 9}};

        for (int i = 0; i < matrix.length; i++) //row

        {

            int sum = 0;  
            for (int j = 0; j < matrix[i].length; j++) //column

            {

                System.out.print(matrix[i][j] + " ");

                sum += matrix[i][j];

            }

            System.out.println("Sum of row " + (i + 1) + ": " + sum);

        }

    }

}

Lab: Write a Java program to initialize and display 2D array elements with sum of each row.

public class App {

    public static void main(String[] args) throws Exception {

        int[][] matrix = new int[][]{{1, 2, 3}, {4, 5, 6, 7}, {8, 9}};

        for (int i = 0; i < matrix.length; i++) // row

        {

            int sum = 0; // Variable to store sum of elements in each row

            for (int j = 0; j < matrix[i].length; j++) // column

            {

                System.out.print(matrix[i][j] + " ");

                sum += matrix[i][j]; // Add the current element to the sum

            }

            System.out.println("Sum of row " + (i + 1) + ": " + sum); // Print sum of the current row

        }

    }

}

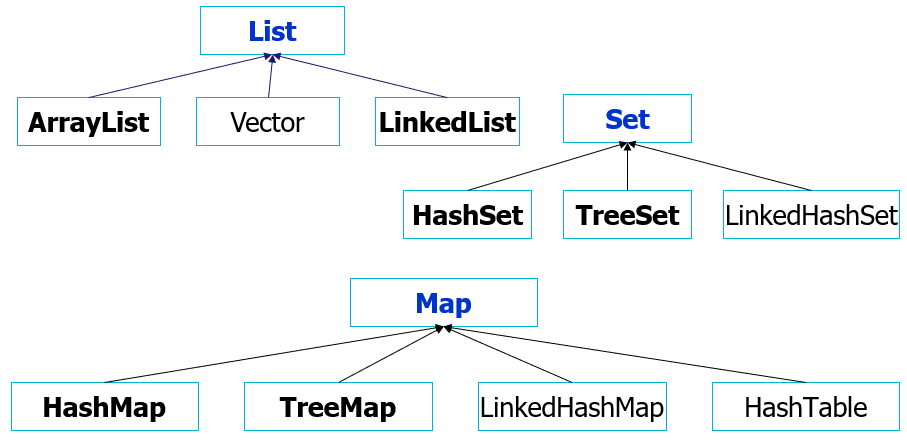
**Collection** is a container of Objects, it groups many Objects into a single one.

**Collections** – dynamic arrays, linked lists, trees, sets, hash tables, stacks, queues.

All collections frameworks contain the following:

* interfaces
* implementations
* algorithms (there are the methods such as searching and sorting)

**Collection in java**



**Interfaces:**

**List:**

* Allows duplicate elements.
* Maintains the order of insertion.
* Elements can be accessed by their index.
* Common implementations include **ArrayList, LinkedList etc**.

**Set:**

* Does not allow duplicate elements.
* Does not maintain any specific order.
* Provides methods to add, remove, and check containment of elements.
* Common implementations include **HashSet and TreeSet**.
* Useful when you need a collection that guarantees uniqueness of elements.

**Map:**

* Stores key-value pairs.
* Keys are unique; duplicate keys are not allowed.
* Provides methods to put, get, and remove elements based on keys.
* Common implementations include **HashMap, TreeMap** etc.

**List:**

Since List is an **interface** you need to instantiate a concrete implementation of the interface in order to use it.

**There are two general-purpose List implementations** — **ArrayList** and **LinkedList**

For example

* java.util.ArrayList
* java.util.LinkedList

**Here are a few examples of how to create a List instance:**

   List listArr = new ArrayList();

   List listLink = new LinkedList();

import java.util.\*;

public class App {

    public static void main(String[] args) throws Exception {

       // Create an ArrayList of integers

       List<Integer> numbers = new ArrayList<>();

       // Adding elements to the ArrayList

       numbers.add(10);

       numbers.add(20);

       numbers.add(30);

       numbers.add(40);

       // Accessing elements by index

       int firstNumber = numbers.get(0);

       System.out.println("First number: " + firstNumber);

       // Iterating through the ArrayList

       System.out.println("All numbers:");

       for (int number : numbers) {

           System.out.println(number);

       }

       // Checking if an element exists in the ArrayList

       boolean contains30 = numbers.contains(30);

       System.out.println("Contains 30: " + contains30);

       // Removing an element from the ArrayList

       boolean removed20 = numbers.remove(Integer.valueOf(20)); // Removing by value

       System.out.println("Removed 20: " + removed20);

       // Clear the ArrayList

       numbers.clear();

       System.out.println("Cleared the ArrayList");

}

}

**ArrayList:**

**Basic Operations on ArrayList**

The ArrayList class provides various methods to perform different operations on arraylists. We will look at some commonly used arraylist operations in this tutorial:

* Add elements
* Access elements
* Change elements
* Remove elements

Example:

import java.util.\*;

public class App {

    public static void main(String[] args) throws Exception {

        // Create an ArrayList of strings

        List<String> names = new ArrayList<>();

        // Adding elements to the ArrayList

        names.add("Nepal");

        names.add("China");

        names.add("India");

        names.add("Pakistan");

        // Accessing elements by index

        String firstCountry = names.get(0);

        System.out.println("First Country: " + firstCountry);

        // Iterating through the ArrayList

        System.out.println("All Country:");

        for (String name : names) {

            System.out.println(name);

        }

        // Checking if an element exists in the ArrayList

        boolean containsBangladesh = names.contains("Bangladesh");

        System.out.println("Contains Bangladesh: " + containsBangladesh);

        // Removing an element from the ArrayList

        boolean removedIndia = names.remove("India");

        System.out.println("Removed India: " + removedIndia);

        // Clear the ArrayList

        names.clear();

        System.out.println("Cleared the ArrayList");

}

}

**Iterate through an ArrayList:**

   // creating an array list

    ArrayList<String> animals = new ArrayList<>();

    animals.add("Cow");

    animals.add("Cat");

    animals.add("Dog");

    System.out.println("ArrayList: " + animals);

    // iterate using for-each loop

    System.out.println("Accessing individual elements:  ");

    for (String language : animals) {

      System.out.print(language);

      System.out.print(", ");

**Lab: Write java program to fetch all items from Arraylist using iterator.**

**Iterating ArrayList using Iterator:**

ArrayList<String> list=new ArrayList<String>();//Creating arraylist

  list.add("Mango");//Adding object in arraylist

  list.add("Apple");

  list.add("Banana");

  list.add("Grapes");

  //Traversing list through Iterator

  Iterator itr=list.iterator();//getting the Iterator

  while(itr.hasNext()){//check if iterator has the elements

   System.out.println(itr.next());//printing the element and move to next

    }

**Linked List:**

**LinkedList** has the same functionality as the ArrayList.

Different way of implementation and efficiency of operations.

* + - Adding to the LinkedList is faster
    - Pass through the list is almost as effective as the ArrayList,
    - Arbitrary(random) removal from the list is slower than ArrayList.

Example:

List<Integer> numbers = new LinkedList<>();

        // Add elements to the linked list

        numbers.add(10);

        numbers.add(20);

        numbers.add(30);

        // Access elements by index

        int firstNumber = numbers.get(0);

        int secondNumber = numbers.get(1);

        // Print the accessed elements

        System.out.println("First number: " + firstNumber);

        System.out.println("Second number: " + secondNumber);

        // Remove an element

        numbers.remove(0);

        // Check if the list contains a specific element

        boolean contains30 = numbers.contains(30);

        System.out.println("Contains 30: " + contains30);

        // Get the size of the linked list

        int size = numbers.size();

        System.out.println("Size of the linked list: " + size);

        // Iterate over the linked list using a for-each loop

        System.out.println("Numbers in the linked list:");

        for (int number : numbers) {

            System.out.println(number);

        }

**Set:**

* A Set is a collection that does not contain any duplicate element.
* Element that are put in a set must override equals() method to establish uniqueness
* It is unsorted, unordered Set
* Can contain null

Example:

import java.util.\*;

public class App {

    public static void main(String[] args) throws Exception {

        Set<String> fruits = new HashSet<>();

        // Adding elements to the set

        fruits.add("Apple");

        fruits.add("Banana");

        fruits.add("Orange");

        fruits.add("Apple"); // Duplicate elements are not allowed

        // Display the contents of the set

        System.out.println("Set contains:");

        for (String fruit : fruits) {

            System.out.println(fruit);

        }

        // Check if an element exists in the set

        boolean containsBanana = fruits.contains("Banana");

        System.out.println("Contains Banana: " + containsBanana);

        // Remove an element from the set

        boolean removed = fruits.remove("Orange");

        System.out.println("Removed Orange: " + removed);

        // Check if the set is empty

        boolean isEmpty = fruits.isEmpty();

        System.out.println("Is the set empty: " + isEmpty);

        // Size of the set

        int setSize = fruits.size();

        System.out.println("Size of the set: " + setSize);

        // Clear the set

        fruits.clear();

        System.out.println("Cleared the set");

}

}

**Map:**

* The most commonly used Map implementations are HashMap and TreeMap.

Map mapA = **new** HashMap();

Map mapB = **new** TreeMap();

import java.util.\*;

public class App {

    public static void main(String[] args) throws Exception {

        // Create a HashMap with Integer keys and String values

        Map<Integer, String> studentMap = new HashMap<>();

        // Adding key-value pairs to the map

        studentMap.put(101, "Nepal");

        studentMap.put(102, "China");

        studentMap.put(103, "India");

        studentMap.put(104, "Pakistan"); // Duplicate values are allowed

        // Display the contents of the map

        System.out.println("Map contains:");

        for (Map.Entry<Integer, String> entry : studentMap.entrySet()) {

        System.out.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());

        }

        // Check if a key exists in the map

        boolean containsKey = studentMap.containsKey(102);

        System.out.println("Contains key 102: " + containsKey);

        // Check if a value exists in the map

        boolean containsValue = studentMap.containsValue("China");

        System.out.println("Contains value 'David': " + containsValue);

        // Get the value associated with a specific key

        String valueForKey = studentMap.get(101);

        System.out.println("Value for key 101: " + valueForKey);

        // Remove a key-value pair from the map

        String removedValue = studentMap.remove(103);

        System.out.println("Removed value for key 103: " + removedValue);

        // Clear the map

        studentMap.clear();

        System.out.println("Cleared the map");

}

}

**Loop in Map:**

     for (Map.Entry<String, String> entry : map.entrySet())

{

      System.out.println(entry.getKey() + " "  + entry.getValue());

     }

Or,

    for (Iterator i = map.entrySet().iterator(); i.hasNext();)

    {

    Map.Entry entry = (Map.Entry) i.next();

    System.out.println(entry.getKey() + " " + entry.getValue());

    }

**Benefits of collection**

* reduces programming effort
* increases program speed and quality
* allows interoperability among unrelated APIs
* reduce effort to design new APIs
* helps to reuse the code