

**Java Collections Implementation**

Contents

[Overview 3](#_Toc193027253)

[Array 3](#_Toc193027254)

[Steps. 3](#_Toc193027255)

[Example: 3](#_Toc193027256)

[Dynamic Array 6](#_Toc193027257)

[Steps. 6](#_Toc193027258)

[Example: 6](#_Toc193027259)

[Linked List 8](#_Toc193027260)

[Steps. 8](#_Toc193027261)

[Example: 8](#_Toc193027262)

[Map 10](#_Toc193027263)

[Steps. 10](#_Toc193027264)

[Example: 10](#_Toc193027265)

[Stack 12](#_Toc193027266)

[Steps. 12](#_Toc193027267)

[Example: 12](#_Toc193027268)

[Queue 14](#_Toc193027269)

[Steps. 14](#_Toc193027270)

[Example: 14](#_Toc193027271)

# Overview

This document is intended to provide how to implement Array, Dynamic Array, Linked List,Stack,Queue and Map in java.

# Array

## Steps.

1-Declare an Array

String[] names;

2-Allocate Memory for the Array

names = new String[4];

3-Initialize the Array

String[] names = { John, Jane, Jack, Bob};

4-Access Array Elements

for (int i = 0; i < names.length; i++) {

System.out.println(names[i]);

}

5-Operations on array(Add,Delete,Update,Search)

Illustrated with an example in the next section

## Example:

**Java Code**

**import java.util.Arrays;**

**public class StudensArrayExample {**

**public static void main(String[] args) {**

**//Declare an Array**

**String[] names;**

**//Allocate Memory for the Array**

**names = new String[4];**

**//Initialize the Array**

**names[0] = "John";**

**names[1] = "Jane";**

**names[2] = "Jack";**

**names[3] = "Bob";**

**//Access Array Elements**

**for (int i = 0; i < names.length; i++) {**

**System.out.println(names[i]);**

**}**

**System.out.println("Original Array: " + Arrays.toString(names));**

**//Adding element to array**

**// New element to add**

**String newElement = "samual";**

**// Create a new array with size 1 more than the original array**

**String[] newArray = new String[names.length + 1];**

**//Copy the original array's elements into the new array**

**for (int i = 0; i < names.length; i++) {**

**newArray[i] = names[i];**

**}**

**// Step 3: Add the new element at the last position**

**newArray[names.length] = newElement;**

**System.out.println("New Array After Adding Samual: " + Arrays.toString(newArray));**

**//Removing element from original array**

**int indexToRemove = 2;**

**//Create a new array with one less size**

**newArray = new String[names.length - 1];**

**// Copy elements before the index to remove**

**for (int i = 0, j = 0; i < names.length; i++) {**

**if (i == indexToRemove) {**

**continue; // Skip the element to be removed**

**}**

**newArray[j++] = names[i];**

**}**

**System.out.println("New Array After Removing Jack: " + Arrays.toString(newArray));**

**//Updating an Element in array**

**names[1] = "Peter";**

**System.out.println("New Array After updating Jane with Peter: " + Arrays.toString(names));**

**//Search for element in array**

**String targetElement = "Bob";**

**//Loop through the array to find the target element**

**int index = searchElement(names, targetElement);**

**if (index != -1) {**

**System.out.println("Element " + targetElement + " found at index: " + index);**

**} else {**

**System.out.println("Element " + targetElement + " not found in the array.");**

**}**

**}**

**public static int searchElement(String[] array, String target) {**

**//Loop through the array to find the target element**

**for (int i = 0; i < array.length; i++) {**

**if (array[i] == target) {**

**return i; // Return the index if found**

**}**

**}**

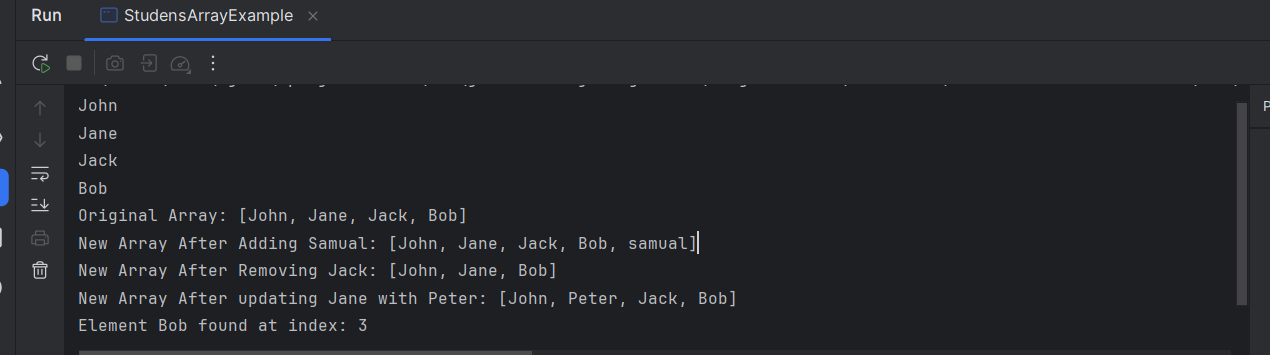
**return -1; // Return -1 if element not found**

**}**

**}**



**Output:**



# Dynamic Array

## Steps.

1- Create an ArrayList: Declare and instantiate an ArrayList object.

2- **Access elements**: Use the get() method to access elements by index.

3**-** **Add elements:** Use the add() method to add elements to the list.

4- **Remove elements**: Use the remove() method to remove elements.

5-Operations on array(Add,Delete,Update,Search)

Illustrated with an example in the next section

## Example:

**Java Code**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**public class DynamicArrayExample {**

**public static void main(String[] args) {**

**//Create a dynamic array using ArrayList**

**ArrayList<String> dynamicArray = new ArrayList<>();**

**//Add elements to the dynamic array**

**dynamicArray.add("John");**

**dynamicArray.add("Jane");**

**dynamicArray.add("Jack");**

**dynamicArray.add("Bob");**

**//Access Array Element**

**for (String element : dynamicArray) {**

**System.out.println(element);**

**}**

**System.out.println("Original Array: " +dynamicArray);**

**//Remove an element (remove element at index 1)**

**dynamicArray.remove(1);**

**System.out.println("New Array After Removing Jane: " + dynamicArray);**

**//Update the element at the specified index using set()**

**dynamicArray.set(1, "Peter");**

**System.out.println("New Array After updating Jack with Peter: " + dynamicArray);**

**//Search for element in array**

**String targetElement = "Bob";**

**if (dynamicArray.contains(targetElement)) {**

**System.out.println("Element " + targetElement + " found at index: " + dynamicArray.indexOf(targetElement));**

**} else {**

**System.out.println("Element " + targetElement + " not found in the array.");**

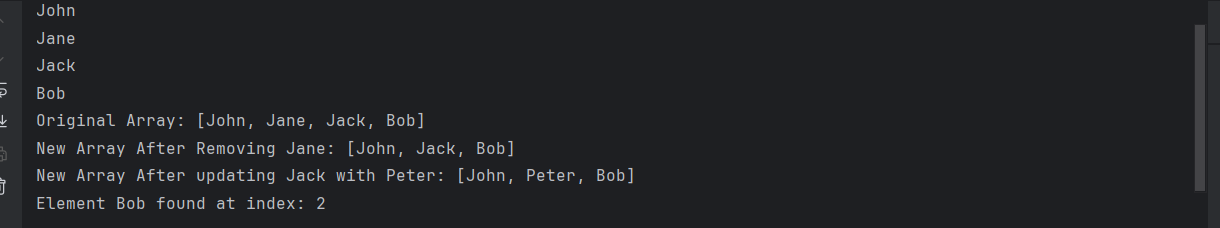
**}**

**}**

**}**



**Output:**



# Linked List

It is a doubly linked list, where each node contains a reference to both the previous and next node.Steps.

You can insert, remove, and access elements from both ends efficiently.

## Steps.

**1-** Adding elements to the LinkedList.

2- Removing elements from the LinkedList.

3 - Check if a specific element exists.

4- Get the size of the list.

## Example:

**Java Code**

**import java.util.LinkedList;**

**public class LinkedListExample {**

**public static void main(String[] args) {**

**LinkedList<String> list = new LinkedList<String>();**

**// Add elements to the LinkedList**

**list.add("John");**

**list.add("Jane");**

**list.add("Jack");**

**list.add("Bob");**

**System.out.println("Original List: " + list);**

**// Add elements at the beginning of the list**

**list.addFirst("Peter");**

**// Add elements at the end of the list**

**list.addLast("Charlie");**

**System.out.println("List after adding peter at first and charlie at end: " + list);**

**//Remove element from the list**

**list.remove(1);**

**System.out.println("List after removing John: " + list);**

**//Remove the first and last elements**

**list.removeFirst(); // Removes John (first element)**

**list.removeLast(); // Removes Charlie (last element)**

**System.out.println("List after removing first and last elements: " + list);**

**// Get the first and last elements without removing**

**String firstElement = list.getFirst(); // Returns the first element (John)**

**String lastElement = list.getLast(); // Returns the last element (Bob)**

**System.out.println("First element: " + firstElement);**

**System.out.println("Last element: " + lastElement);**

**// Check if a specific element exists**

**String targetElement = "Bob";**

**if(list.contains(targetElement)) {**

**System.out.println("Element " + targetElement + " found at index: " + list.indexOf(targetElement));**

**}else{**

**System.out.println("Element " + targetElement + " not found");**

**}**

**// Get the size of the list**

**int size = list.size();**

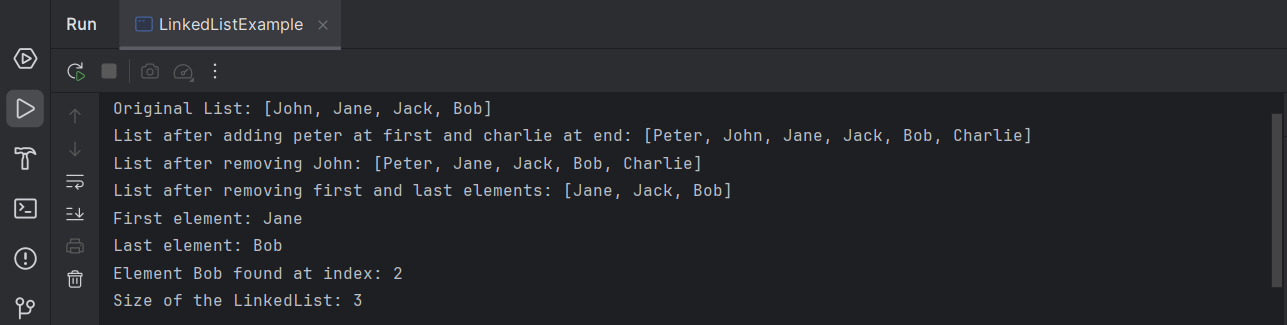
**System.out.println("Size of the LinkedList: " + size);**

**}**

**}**



**Output:**



# Map

In Java, a Map is part of the java.util package and represents an object that maps keys to values.

Here’s a basic guide on how to implement and use a Map in Java, specifically using HashMap as an example:

## Steps.

**1-** Adding elements to the Map

2- Accessing elements in the Map

3 Removing elements from the Map.

4- Checking if a key or value exists

5- Iterating over the Map

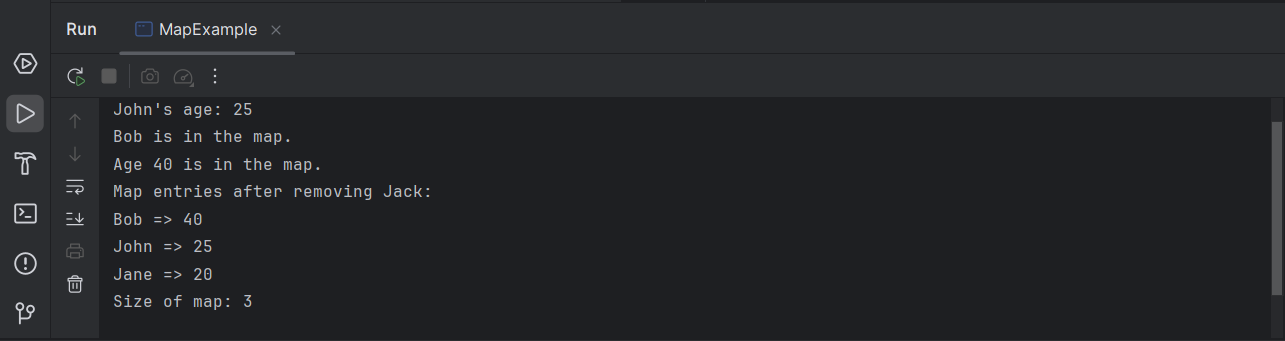
## Example:

**Java Code**

**import java.util.HashMap;  
import java.util.Map;  
  
public class MapExample {  
 public static void main(String[] args) {  
 // Create a Map with String keys and Integer values  
 Map<String, Integer> map = new HashMap<>();  
  
 // Add elements to the map  
 map.put("John", 25);  
 map.put("Jane", 20);  
 map.put("Jack", 30);  
 map.put("Bob", 40);  
  
 // Access a value by key  
 System.*out*.println("John's age: " + map.get("John"));  
  
 // Check if a key exists  
 if (map.containsKey("Bob")) {  
 System.*out*.println("Bob is in the map.");  
 }  
  
 // Check if a value exists  
 if (map.containsValue(40)) {  
 System.*out*.println("Age 40 is in the map.");  
 }  
  
 // Remove an element  
 map.remove("Jack");  
  
 // Iterate through the map  
 System.*out*.println("Map entries after removing Jack:");  
 for (Map.Entry<String, Integer> entry : map.entrySet()) {  
 System.*out*.println(entry.getKey() + " => " + entry.getValue());  
 }  
  
 // Print size of map  
 System.*out*.println("Size of map: " + map.size());  
 }  
}**



**Output:**



# Stack

A **Stack** is a data structure that follows the **Last In, First Out (LIFO)** principle. In a stack, the last element added is the first one to be removed.

## Steps.

1- **Push**: Adds an element to the top of the stack.

2- **Pop**: Removes the top element from the stack.

3 **Peek**: Returns the top element without removing it.

4- **isEmpty**: Checks if the stack is empty.

5- **Size**: Returns the size of the stack.

## Example:

**Java Code**

**import java.util.Stack;**

**public class StackExample {**

**public static void main(String[] args) {**

**// Create a stack with a capacity of 4**

**Stack<String> stack = new Stack<String>();**

**// Push elements to the stack**

**stack.push("John");**

**stack.push("Jane");**

**stack.push("Jack");**

**stack.push("Bob");**

**System.out.println("Original Stack" + stack);**

**// Peek the top element**

**System.out.println("Top element is: " + stack.peek());**

**// Pop elements from the stack**

**stack.pop(); // Output: John popped from stack**

**stack.pop(); // Output: Jane popped from stack**

**System.out.println("Stack After pop Jack, Bob " + stack);**

**// Check the size of the stack**

**System.out.println("Current stack size: " + stack.size());**

**// Check if the stack is empty**

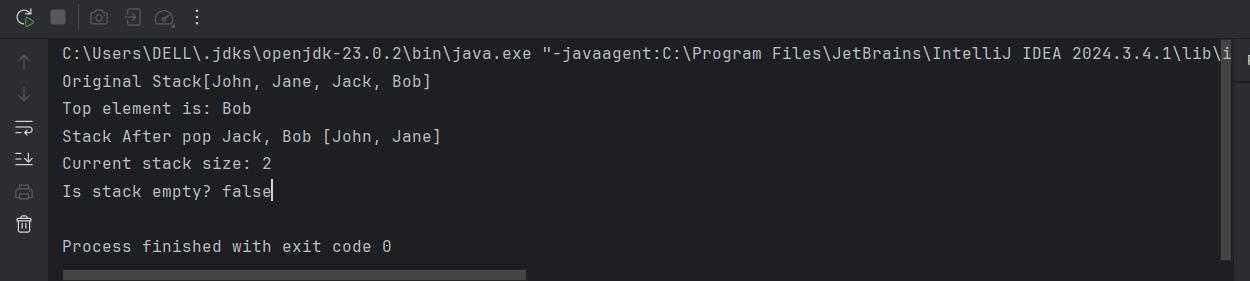
**System.out.println("Is stack empty? " + stack.isEmpty());**

**}**

**}**



**Output:**



# Queue

A Queue is a data structure that follows the First In, First Out (FIFO) principle, meaning the first element added to the queue is the first one to be removed.

## Steps.

**1-** **offer**: Adds an element to the end of the queue.

2- **poll**: Retrieves and removes the head of the queue, or returns null if the queue is empty.

3 **Peek**: Retrieves, but does not remove, the head of the queue, or returns null if the queue is empty.

4- **isEmpty**: Returns true if the queue contains no elements.

5- **Size**: Returns the number of elements in the queue.

## Example:

**Java Code**

**import java.util.LinkedList;**

**import java.util.Queue;**

**public class QueueExample {**

**public static void main(String[] args) {**

**// Create a queue using LinkedList (LinkedList implements Queue)**

**Queue<String> queue = new LinkedList<>();**

**// Enqueue elements using offer() method**

**queue.offer("John");**

**queue.offer("Jane");**

**queue.offer("Jack");**

**queue.offer("Bob");**

**System.out.println("Original Queue" + queue);**

**// Peek the front element of the queue using peek() method**

**System.out.println("Front element is: " + queue.peek()); // Output: Front element is: John**

**// Dequeue elements using poll() method (removes from the front)**

**queue.poll(); // Output: Dequeued: John**

**queue.poll(); // Output: Dequeued: Jane**

**System.out.println("Queue After poll John, Jane " + queue);**

**// Check the size of the queue**

**System.out.println("Queue size: " + queue.size()); // Output: Queue size: 1**

**// Check if the queue is empty using isEmpty()**

**System.out.println("Is queue empty? " + queue.isEmpty());**

**}**

**}**



**Output:**

