### **Java ArrayList**

### ArrayList in Java is a part of the java.util package. It is a resizable array, which means elements can be added or removed dynamically.

#### **1. Creating an ArrayList**

To create an ArrayList, you need to import java.util.ArrayList and then instantiate it.

**Example:**

import java.util.ArrayList;

public class ArrayListExample {

public static void main(String[] args) {

// Creating an ArrayList of integers

ArrayList<Integer> intList = new ArrayList<>();

// Creating an ArrayList of strings

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

}

}

#### **2. Inserting Values**

You can add values using the add method.

**Example:**

import java.util.ArrayList;

public class ArrayListExample {

public static void main(String[] args) {

ArrayList<Integer> intList = new ArrayList<>();

intList.add(10); // Adding an integer

intList.add(20);

intList.add(30);

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

stringList.add("Hello"); // Adding a string

stringList.add("World");

// Printing the ArrayLists

System.out.println("Integer List: " + intList);

System.out.println("String List: " + stringList);

}

}

#### **3. Deleting Values**

You can remove values using the remove method by index or by value.

**Example:**

import java.util.ArrayList;

public class ArrayListExample {

public static void main(String[] args) {

ArrayList<Integer> intList = new ArrayList<>();

intList.add(10);

intList.add(20);

intList.add(30);

// Removing value by index

intList.remove(1); // Removes the element at index 1 (20)

// Removing value by value (first occurrence)

intList.remove(Integer.valueOf(30));

// Printing the updated list

System.out.println("Updated Integer List: " + intList);

}

}

#### **4. Reversing an ArrayList**

You can reverse an ArrayList using Collections.reverse method.

**Example:**

import java.util.ArrayList;

import java.util.Collections;

public class ArrayListExample {

public static void main(String[] args) {

ArrayList<Integer> intList = new ArrayList<>();

intList.add(10);

intList.add(20);

intList.add(30);

// Reversing the list

Collections.reverse(intList);

// Printing the reversed list

System.out.println("Reversed Integer List: " + intList);

}

}

#### **5. Sorting an ArrayList**

You can sort an ArrayList in ascending order using Collections.sort method.

**Example:**

import java.util.ArrayList;

import java.util.Collections;

public class ArrayListExample {

public static void main(String[] args) {

ArrayList<Integer> intList = new ArrayList<>();

intList.add(30);

intList.add(10);

intList.add(20);

// Sorting the list in ascending order

Collections.sort(intList);

// Printing the sorted list

System.out.println("Sorted Integer List: " + intList);

}

}

**9.** The ArrayList class does not have a built-in method named merge. However, you can merge two ArrayLists manually by using the **addAll()** method or iterating over one list and adding its elements to another.

Here’s how you can merge two ArrayLists in Java:

### **Using addAll()**

The addAll() method adds all the elements from one list to another.

**Example:**

import java.util.ArrayList;

public class MergeExample {

public static void main(String[] args) {

ArrayList<Integer> list1 = new ArrayList<>();

list1.add(1);

list1.add(2);

list1.add(3);

ArrayList<Integer> list2 = new ArrayList<>();

list2.add(4);

list2.add(5);

list2.add(6);

// Merging list2 into list1

list1.addAll(list2);

// Printing the merged list

System.out.println("Merged List: " + list1);

}

}

**ADDITIONAL**

### **ArrayList Methods**

#### **1. Adding Elements**

* **add(E e)**: Adds an element at the end.
* **add(int index, E e)**: Inserts an element at the specified index.

#### **2. Accessing Elements**

* **get(int index)**: Retrieves the element at the specified index.

ArrayList<Integer> list = new ArrayList<>();

list.add(10);

int element = list.get(0); // element = 10

#### **3. Updating Elements**

* **set(int index, E element)**: Replaces the element at the specified index with the specified element.

list.set(0, 20); // Replaces the value at index 0 with 20

#### **4. Removing Elements**

* **remove(int index)**: Removes the element at the specified index.
* **remove(Object o)**: Removes the first occurrence of the specified element.
* **clear()**: Removes all elements from the list

list.clear(); // Empties the list

#### **5. Searching Elements**

* **contains(Object o)**: Checks if the list contains the specified element.

boolean exists = list.contains(10); // true if 10 exists

**indexOf(Object o)**: Returns the index of the first occurrence of the element, or -1 if not found.

**lastIndexOf(Object o)**: Returns the index of the last occurrence of the element.

#### 

#### **6. Size and Capacity**

* **size()**: Returns the number of elements in the list.

int size = list.size(); // Gets the number of elements

**ensureCapacity(int minCapacity)**: Ensures the capacity is at least the specified size.

**trimToSize()**: Reduces the storage capacity of the ArrayList to its current size.

#### **7. Iterating Through the List**

* **forEach(Consumer<? super E> action)**: Performs the specified action for each element.

list.forEach(System.out::println); // Prints each element

**iterator()**: Returns an iterator for the list.

**listIterator()**: Returns a list iterator to traverse the list in both directions.

#### **8. Sublist and Views**

* **subList(int fromIndex, int toIndex)**: Returns a view of the portion of the list between the specified indices.

List<Integer> sublist = list.subList(0, 2); // Sublist of first two elements

#### **9. Comparing Lists**

* **equals(Object o)**: Compares the list with another object for equality.

#### **10. Other Utility Methods**

* **isEmpty()**: Checks if the list is empty

boolean empty = list.isEmpty(); // true if the list is empty

Object[] array = list.toArray();

**toArray(T[] a)**: Converts the list into the specified type of array.