

**Lab Manual- ArrayList , LinkedList and HashMap**

**Prepared for**:

**Date:** 18th June 2024

**Prepared by:**

Document Name: Lab Manual **Document Number** SDLab333

**Contributor:**

Contents

[1. ArrayList and LinkedList 3](#_Toc172448770)

[2. ArrayList 3](#_Toc172448771)

[3. LinkedList 4](#_Toc172448772)

[4. ArrayList and LinkedList Example 5](#_Toc172448773)

[5. HashMap List and Method 7](#_Toc172448774)

[5.1 Key Concepts 7](#_Toc172448775)

[5.2 Real-Life Analogy: Phone Directory 7](#_Toc172448776)

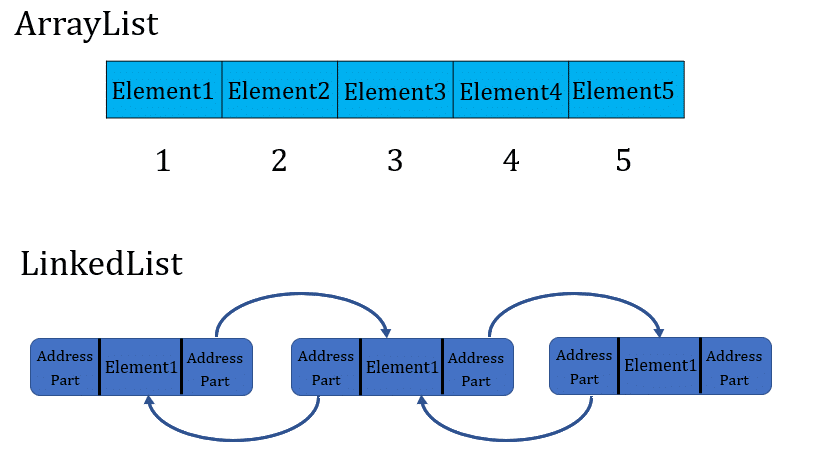
[5.3 How It Works 8](#_Toc172448777)

[5.4 Basic Map Methods Example 8](#_Toc172448778)

[1. Explanation of Basic Map Methods 9](#_Toc172448779)

# ArrayList and LinkedList

In java,**[ArrayList](https://javagoal.com/arraylist-in-java/" \t "_blank)and**[**LinkedList**](https://javagoal.com/java-linkedlist/)both are linear data structures in the [**Collection framework**](https://javagoal.com/collections-in-java/). Both data structures introduced due to the limitation of the array because the **Array has a predefined and fixed size**. In this post, we will see the **difference between ArrayList and LinkedList.**There are many similarities in both, but we will discuss how **ArrayList vs LinkedList** in deep.



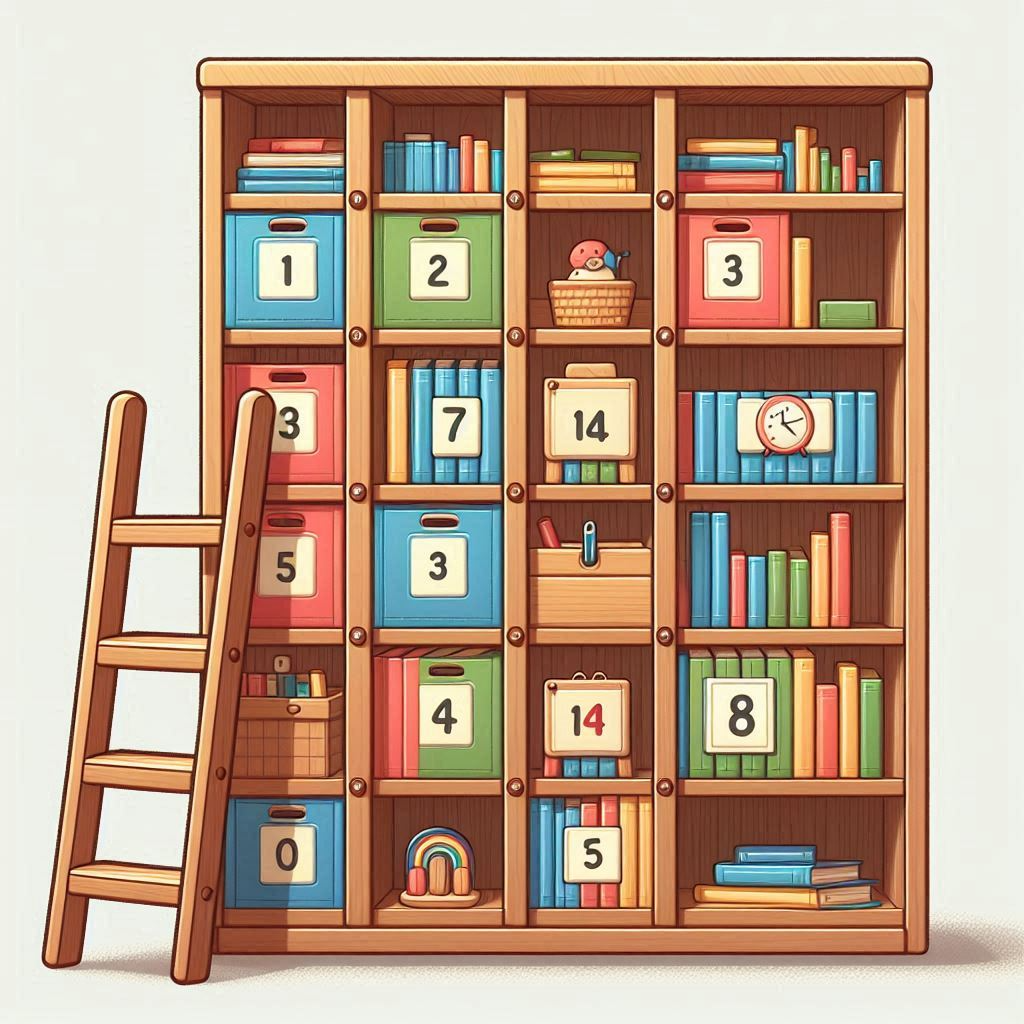
# ArrayList

**Scenario: Books on a Shelf**

Imagine you have a bookshelf with numbered slots. Each slot can hold exactly one book. If you want to add a book, you simply put it in the next available slot. If you want to remove a book, you take it out and leave an empty slot or move all the subsequent books one slot forward to fill the gap.

**Characteristics:**

* **Fixed Slot Positions:** Each slot on the shelf is numbered, just like array indices.
* **Direct Access:** You can quickly access any book by its slot number.
* **Resizing:** If the shelf is full and you want to add more books, you need a bigger shelf. This involves creating a new, larger shelf and moving all the books to the new shelf.

****

# LinkedList

**Scenario: Treasure Hunt with Linked Clues**

Imagine a treasure hunt where each clue leads you to the next clue. You start with the **first clue**, **which tells you where to find the second clue**, and so on until you find the treasure.

**Characteristics:**

* **Sequential Access:** You follow the clues in sequence. You can't skip directly to a specific clue without following the chain from the start.
* **Flexibility:** Adding or removing a clue is easy. You simply adjust the link from the previous clue to point to the new clue or skip the removed clue.
* **Dynamic Size:** You can add or remove clues without worrying about predefined slots or space.

****

# ArrayList and LinkedList Example

**ArrayListExample.java**

import java.util.ArrayList;

public class ArrayListExample {

    public static void main(String[] args) {

        // Create an ArrayList

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

        // Add elements to the ArrayList

        list.add("Apple");

        list.add("Banana");

        list.add("Orange");

        // Display the ArrayList

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

        // Get an element from the ArrayList

        String fruit = list.get(1);

        System.out.println("Element at index 1: " + fruit);

        // Remove an element from the ArrayList

        list.remove("Banana");

        System.out.println("ArrayList after removal: " + list);

        // Iterate through the ArrayList

        System.out.print("Iterating through ArrayList: ");

        for (String item : list) {

            System.out.print(item + " ");

        }

    }

}

**LinkedListExample.java**

import java.util.LinkedList;

public class LinkedListExample {

    public static void main(String[] args) {

        // Create a LinkedList

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

        // Add elements to the LinkedList

        list.add("Dog");

        list.add("Cat");

        list.add("Rabbit");

        // Display the LinkedList

        System.out.println("LinkedList: " + list);

        // Get an element from the LinkedList

        String animal = list.get(1);

        System.out.println("Element at index 1: " + animal);

        // Remove an element from the LinkedList

        list.remove("Cat");

        System.out.println("LinkedList after removal: " + list);

        // Iterate through the LinkedList

        System.out.print("Iterating through LinkedList: ");

        for (String item : list) {

            System.out.print(item + " ");

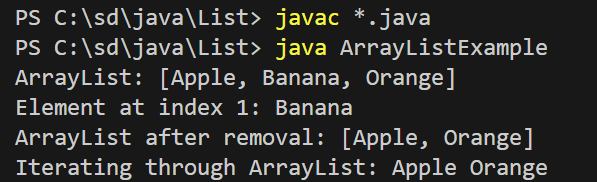
        }

    }

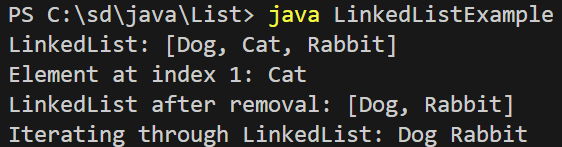
}

**javac \*.java**

**java ArrayListExample**

****

**java LinkedListExample**

****

# HashMap List and Method

In simple terms, a HashMap is like a digital version of a physical dictionary or phone book. It is a data structure that stores data in key-value pairs, allowing you to quickly look up a value (like a phone number) using its associated key (like a person's name).

## Key Concepts

1. **Key-Value Pairs**:
   * **Key:** A unique identifier (like a name in a phone book).
   * **Value:** The data associated with the key (like a phone number).
2. **Hashing**:
   * The HashMap uses a process called hashing to convert the key into a unique code (called a hash code) which determines where the key-value pair is stored in the map.
3. **Fast Lookups**:
   * Due to hashing, HashMap allows for very fast retrieval of values when you know the key.

 **Map** is a collection that maps keys to values.

 **HashMap** is a common implementation of the Map interface.

 Basic operations include adding (put), accessing (get), removing (remove), and checking for keys (containsKey).

 You can iterate through the map using the entrySet() method.

## Real-Life Analogy: Phone Directory

Imagine you have a phone directory:

* Each entry in the directory has a person's name (key) and their phone number (value).
* You can quickly find someone's phone number if you know their name.

## How It Works

* **Adding Entries:** You add entries to the HashMap using the key-value pairs. For example, you can add "John" with his phone number "123-456-7890".
* **Retrieving Entries:** If you want to find John's phone number, you just need to look up "John" in the HashMap, and it will quickly give you "123-456-7890".
* **Removing Entries:** If John changes his phone number, you can remove his old entry and add a new one.

## Basic Map Methods Example

Here's a very simple example to understand the concept of MapView and basic map methods in Java.

**Scenario: Phone Directory**

Imagine you have a phone directory where you store the phone numbers of people. Each entry in the phone directory has a person's name (the key) and their phone number (the value).

Here is a simple example in Java using HashMap:

**MapExample.java**

import java.util.HashMap;

import java.util.Map;

public class MapExample {

    public static void main(String[] args) {

        // Create a HashMap

        Map<String, String> phoneDirectory = new HashMap<>();

        // Adding entries to the HashMap

        phoneDirectory.put("John", "123-456-7890");

        phoneDirectory.put("Jane", "987-654-3210");

        phoneDirectory.put("Bob", "555-666-7777");

        // Display the HashMap

        System.out.println("Phone Directory: " + phoneDirectory);

        // Accessing an element from the HashMap

        String johnNumber = phoneDirectory.get("John");

        System.out.println("John's number: " + johnNumber);

        // Removing an entry from the HashMap

        phoneDirectory.remove("Jane");

        System.out.println("Phone Directory after removal: " + phoneDirectory);

        // Checking if a key exists in the HashMap

        boolean hasBob = phoneDirectory.containsKey("Bob");

        System.out.println("Does Bob exist in the directory? " + hasBob);

        // Iterating through the HashMap

        System.out.println("Iterating through the phone directory:");

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

            System.out.println("Name: " + entry.getKey() + ", Phone Number: " + entry.getValue());

        }

    }

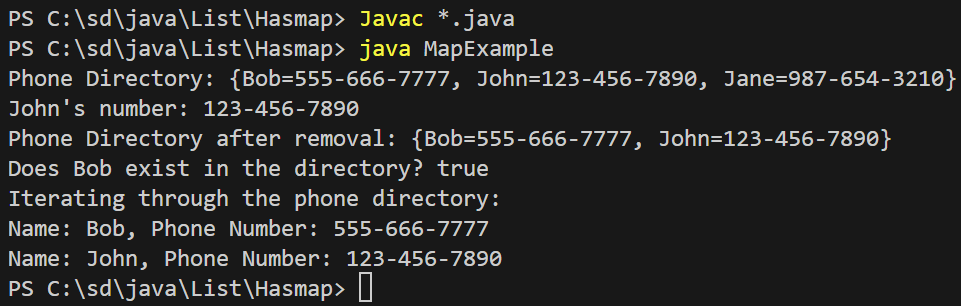
}

### Explanation of Basic Map Methods

1. **put(K key, V value)**
   * **Example:** phoneDirectory.put("John", "123-456-7890");
   * **Explanation:** Adds a key-value pair to the map. If the key already exists, the old value is replaced with the new value.
2. **get(Object key)**
   * **Example:** String johnNumber = phoneDirectory.get("John");
   * **Explanation:** Returns the value to which the specified key is mapped, or null if the map contains no mapping for the key.
3. **remove(Object key)**
   * **Example:** phoneDirectory.remove("Jane");
   * **Explanation:** Removes the mapping for a key from this map if it is present.
4. **containsKey(Object key)**
   * **Example:** boolean hasBob = phoneDirectory.containsKey("Bob");
   * **Explanation:** Returns true if this map contains a mapping for the specified key.

**javac \*.java**

**java MapExample**

****