JAVA COLLECTION (LIST INTERFACE)

The List Interface in Java extends the Collection Interface and is a part of java.util package. It is used to store the ordered collections of elements. So in a Java List, you can organize and manage the data sequentially.

Java List – Operations

Since List is an interface, it can be used only with a class that implements this interface. Now, let's see how to perform a few frequently used operations on the List.

Operation 1: Adding elements to List using add() method

Operation 2: Updating elements in List using set() method

Operation 3: Searching for elements using indexOf(), lastIndexOf methods

Operation 4: Removing elements using remove() method

Operation 5: Accessing Elements in List using get() method

Operation 6: Checking if an element is present in the List using contains() method

Complexity of List Interface in Java

Operation	Time Complexity	Space Complexity
Adding Element in List Interface	O(1)	O(1)
Remove Element from List Interface	O(N)	O(N)
Replace Element in List Interface	O(N)	O(N)

Operation	Time Complexity	Space Complexity
Traversing List Interface	O(N)	O(N)

1. ARRAY LIST:

- Java ArrayList is a part of collections framework and it is a class of java.util package. It provides us with dynamic sized arrays in Java.
- The main advantage of ArrayList is, unlike normal arrays, we don't need to mention the size when creating ArrayList. It automatically adjusts its capacity as elements are added or removed.
- It may be slower than standard arrays, but helpful when size is not known in advance. Note that creating large fixed sized array would cause wastage of space.
- Since ArrayList is part of collections framework, it has better interoperability with other collections. For example conversion to a HashSet is straightforward.
- With generics, ArrayList<T> ensures type safety at compile-time.

```
// Java Program to demonstrate ArrayList
import java.util.ArrayList;

class Main {
   public static void main (String[] args) {

        // Creating an ArrayList
        ArrayList<Integer> a = new ArrayList<Integer>();

        // Adding Element in ArrayList
        a.add(1);
        a.add(2);
        a.add(3);

        // Printing ArrayList
        System.out.println(a);

   }
```

}

Advantages of Java ArrayList

- Dynamic size: ArrayList can dynamically grow and shrink in size, making it easy to add or remove elements as needed.
- Easy to use: ArrayList is simple to use, making it a popular choice for many Java developers.
- Fast access: ArrayList provides fast access to elements, as it is implemented as an array under the hood.
- Ordered collection: ArrayList preserves the order of elements, allowing you to access elements in the order they were added.
- Supports null values: ArrayList can store null values, making it useful in cases where the absence of a value needs to be represented.

Disadvantages of Java ArrayList

- Slower than arrays: ArrayList is slower than arrays for certain operations, such as inserting elements in the middle of the list.
- Increased memory usage: ArrayList requires more memory than arrays, as it needs to maintain its dynamic size and handle resizing.
- Not thread-safe: ArrayList is not thread-safe, meaning that multiple threads may access and modify the list concurrently, leading to potential race conditions and data corruption.
- Performance degradation: ArrayList's performance may degrade as the number of elements in the list increases, especially for operations such as searching for elements or inserting elements in the middle of the list.

2. LINKED LIST:

LinkedList is a class that is implemented in the collection framework which inherently implements the linked list data structure. It is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element

is known as a node. Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays.

```
// Java program to add elements to a LinkedList
import java.util.LinkedList;

public class AddElements {

   // Main driver method
   public static void main(String[] args) {
      // Creating a LinkedList
      LinkedList<String> l = new LinkedList<String>();

      // Adding elements to the LinkedList using add() method
      l.add("One");
      l.add("Two");
      l.add("Three");
      l.add("Four");
      l.add("Five");

      // Printing the LinkedList
      System.out.println(l);
   }
}
```

Advantages of using LinkedList in Java:

- Dynamic size: As with Vector, the size of a LinkedList can grow or shrink dynamically, so you don't have to worry about setting an initial size.
- Efficient Insertions and Deletions: LinkedList is an efficient data structure for inserting or deleting elements in the middle of the list because you only need to change the links between elements, rather than shifting all elements after the insertion or deletion point.
- Flexible Iteration: With a linked list, you can efficiently iterate through the list in either direction, since each element has a reference to both its predecessor and successor elements.

Disadvantages of using LinkedList in Java

- Performance: LinkedList has a slower performance than ArrayList when it comes to accessing individual elements. This is because you need to traverse the list to reach the desired element, whereas with ArrayList, you can simply access the desired element using an index.
- Memory overhead: LinkedList requires more memory than ArrayList because each element requires additional memory for the links to its predecessor and successor elements.