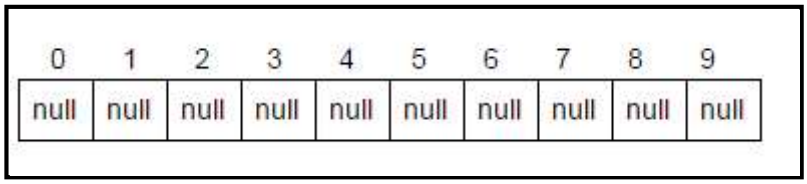
ArrayList custom implementation in java - How ArrayList works internally with diagrams and full program

Contents of page :

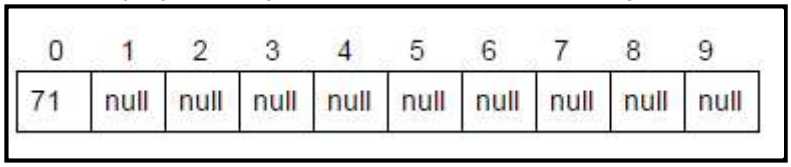
1) Custom ArrayList in java >

In this post i will be explaining ArrayList **Custom** implementation.

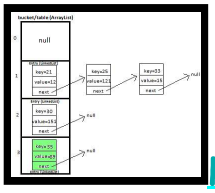
Initially, when we **declare ArrayList<Integer>** with `INITIAL_CAPACITY = 10`, it will be like this-



Let's **add(71)** in ArrayList, after addition our ArrayList will look like this-



Must read:  Find sum of both diagonals in matrix.



HashMap Custom implementation.

2) Methods used in custom ArrayList in java >

public void **add**(E value)	Add objects in **ArrayListCustom**
public E **get**(int index)	Method returns element on specific index.
public Object **remove**(int index)	Method returns removedElement on specific index, else it throws IndexOutOfBoundException if index is negative or greater than size of size.
public void **display**()	-Method displays all objects in **ArrayListCustom**. **-Insertion order is guaranteed.**
private void **ensureCapacity**()	Method increases capacity of list by making it double.

3) Full Program/SourceCode for implementing custom ArrayList in java >

```
import java.util.Arrays;

/**
 * @author AnkitMittal
 * Copyright (c) JavaMadeSoEasy.com , AnkitMittal .
 * All Contents are copyrighted and must not be reproduced in any form.
 * This class provides custom implementation of ArrayList(without using java api's)
 * Insertion order of objects is maintained.
 * Implementation allows you to store null as well.
 * @param <E>
 */
class ArrayListCustom<E> {

    // Define INITIAL_CAPACITY, size of elements of custom ArrayList
    private static final int INITIAL_CAPACITY = 10;
    private int size = 0;
    private Object elementData[] = {};

    /**
     *
     * constructor of custom ArrayList
     */
```

```
public ArrayListCustom() {
	elementData = new Object[INITIAL_CAPACITY];
}

/**
 * add elements in custom/your own ArrayList
 * Method adds elements in ArrayListCustom.
 */
public void add(E e) {
	if (size == elementData.length) {
		ensureCapacity(); // increase current capacity of list, make it
						// double.
	}
	elementData[size++] = e;
}

/**
 * method returns element on specific index.
 */
@SuppressWarnings("unchecked")
public E get(int index) {
	// if index is negative or greater than size of size, we throw
	// Exception.
	if (index < 0 || index >= size) {
		throw new IndexOutOfBoundsException("Index: " + index + ", Size "
				+ index);
	}
	return (E) elementData[index]; // return value on index.
}

/**
 * remove elements from custom/your own ArrayList method returns
 *
 * removedElement on specific index. else it throws IndexOutOfBoundException
 * if index is negative or greater than size of size.
 */
public Object remove(int index) {
	// if index is negative or greater than size of size, we throw
	// Exception.
    if (index < 0 || index >= size) {
		throw new IndexOutOfBoundsException("Index: " + index + ", Size "
				+ index);
	}

	Object removedElement = elementData[index];
	for (int i = index; i < size - 1; i++) {
		elementData[i] = elementData[i + 1];
	}
	size--; // reduce size of ArrayListCustom after removal of element.

	return removedElement;
```

```
    }

    /**
     * Ensure capacity of custom/your own ArrayList.
     *
     * Method increases capacity of list by making it double.
     */
    private void ensureCapacity() {
           int newIncreasedCapacity = elementData.length * 2;
           elementData = Arrays.copyOf(elementData, newIncreasedCapacity);
    }

    /**
     * Display custom/your own ArrayList.
     *
     * Method displays all the elements in list.
     */
    public void display() {
           System.out.print("Displaying list : ");
           for (int i = 0; i < size; i++) {
                  System.out.print(elementData[i] + " ");
           }
    }

}

/**
 * Main class to test ArrayListCustom functionality.
 */
public class ArrayListCustomApp {

    public static void main(String... a) {
           ArrayListCustom<Integer> list = new ArrayListCustom<Integer>();
           //Add elements in custom ArrayList
           list.add(1);
           list.add(2);
           list.add(3);
           list.add(4);
           list.add(1);
           list.add(2);

           //Display custom ArrayList
           list.display();
           System.out.println("\nelement at index in custom ArrayList > " + 1 + " = " +
list.get(1));

           //Remove element from custom ArrayList
           System.out.println("element removed from index " + 1 + " = "
                        + list.remove(1));

           //Again display custom ArrayList
           System.out
```

```
                .println("\nlet's display custom ArrayList again after removal at index 1");

        list.display();

        // list.remove(11); //will throw IndexOutOfBoundsException, because
                // there is no element to remove on index 11.
        // list.get(11); //will throw IndexOutOfBoundsException, because there
                // is no element to get on index 11.
    }

}

/*OUTPUT

Displaying list : 1 2 3 4 1 2
element at index in custom ArrayList > 1 = 2
element removed from index 1 = 2

let's display custom ArrayList again after removal at index 1
Displaying list : 1 3 4 1 2

*/
```

4) Complexity of methods in ArrayList in java >

Operation/ method	Worst case	Best case
add	O(n), when array is full it needs restructuring, operation runs in *amortized constant time.*	O(1), when array does not need any restructuring.
remove	O(n), when removal is done from between restructuring is needed.	O(1), when removal is done at last position, no restructuring is needed.
get	O(1), it is index based structure. So, complexity of get operation is always done in O(1).	O(1) it is index based structure. So, complexity of get operation is always done in O(1).
display	O(n), because iteration is done over each and every element.	O(n), because iteration is done over each and every element.