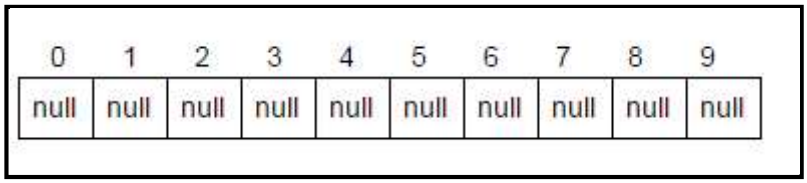
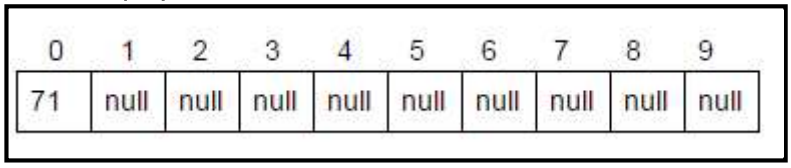
1) Custom Vector in java >

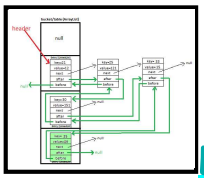
In this post i will be explaining **Vector Custom** implementation.

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



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



Must read:  LinkedHashMap Custom implementation.

Main difference between ArrayList and vector is that vectors can be used in **multithreaded** environment because its method are **synchronized.**

2) Methods used in custom Vector in java >

public **synchronized** void **add**(E value)	Add objects in **VectorCustom**
public E **get**(int index)	Method returns element on specific index.
public **synchronized** 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 **VectorCustom**. **-Insertion order is guaranteed.**
private void **ensureCapacity**()	Method increases capacity of list by making it double.

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

```
package com.ankit;

import java.util.Arrays;

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

  private static final int INITIAL_CAPACITY = 10;
  private Object elementData[]={};
  private int size = 0;

  /**
  * constructor.
  */
  public VectorCustom() {
    elementData = new Object[INITIAL_CAPACITY];
  }

  /**
   * method adds elements in VectorCustom.
   */
  public synchronized 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 synchronized 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.
  }


  /**
```

```
   * method returns removedElement on specific index.
   * else it throws IndexOutOfBoundException if index is negative or greater than size of size.
   */
  public synchronized Object remove(int index) {
    if ( index <0 || index>= size) {  //if index is negative or greater than size of size, we
throw Exception.
      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 VectorCustom after removal of element.

    return removedElement;
  }


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

  /**
   * 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]+" ");
      }
  }

}

/** Copyright (c), AnkitMittal  JavaMadeSoEasy.com */
/**
 * Main class to test VectorCustom functionality.
 */
public class VectorCustomApp {

      public static void main(String...a) {
           VectorCustom<Integer> list = new VectorCustom<Integer>();
     list.add(1);
     list.add(2);
     list.add(3);
     list.add(4);
     list.add(1);
     list.add(2);

     list.display();
     System.out.println("\nelement at index "+1+" = "+list.get(1));
     System.out.println("element removed from index "+1+" = "+list.remove(1));

     System.out.println("\nlet's display list 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 1 = 2
element removed from index 1 = 2

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

*/
```

4) Complexity of methods in Vector 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.